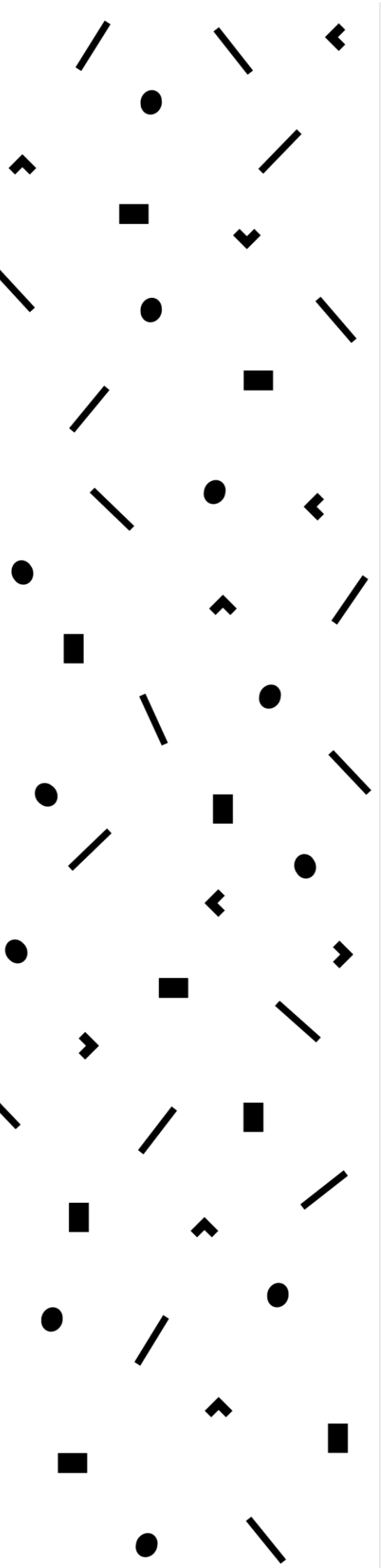


LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

ASIAKASYMPÄRISTÖVALVONNAN AUTOMATISOINTI

Valvontojen ylläpito ja seuranta

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2015
Niki Halm

Lahden ammattikorkeakoulu
Tietotekniikan koulutusohjelma

HALM, NIKI:

Asiakasympäristövalvonnan automatisointi
Valvontojen ylläpito ja seuranta

Ohjelmistotekniikan opinnäytetyö, 45 sivua

Kevät 2015

TIIVISTELMÄ

Tässä opinnäytetyössä tutkittiin asiakasympäristövalvontojen automatisointia. Päivittäisvalvonnat ovat joukko rutiineja, jotka suoritetaan asiakkaan ympäristössä ajastetusti. Valvonnat haluttiin automatisoida nopeamman valvonnan suorittamiseksi, jolloin virhetilanteisiin on nopea puuttua.

Työ tehtiin CGI Suomi Oy:n Lahden toimipisteelle. CGI on viidenneksi suurin it-palveluyritys maailmassa, ja Suomessa se on toiseksi suurin. CGI Suomi Oy työllistää Suomessa noin 3000 henkilöä ja sen tarjoamia palveluita ovat konsultointi, tietojärjestelmien integraatiot, IT- ja liiketoimintaprosessien ulkoistamispalvelut sekä IT-palveluiden toteuttaminen.

Opinnäytetyössä suunniteltiin ja toteutettiin valvontasovellus, jolla voidaan luoda valvontarutiinit ajettavaksi asiakasympäristössä. Valvontasovelluksen tarkoituksena oli nopeuttaa valvontojen suorittamista, kerätä valvontatieja asiakasympäristöistä ja historioida niitä pidempiaikaisraportointia varten.

Asiasanat: tietovarasto, Microsoft Azure -pilvipalvelin, .NET Framework, Windows Communication Foundation, Entity Framework, valvonta

Lahti University of Applied Sciences
Degree Programme in Information Technology

HALM, NIKI:

Automation of Customer Environment Monitoring
Administration and control of monitoring

Bachelor's Thesis in software engineering, 45 pages

Spring 2015

ABSTRACT

The objective of this thesis was to investigate the automation of the monitoring of customer environment. Daily monitoring is a group of routines, which are executed in the customer's environment according to a schedule. The purpose of automation was to make monitoring faster, so potential errors can be intervened.

The work was accomplished for CGI Finland Ltd, for it's Lahti office. CGI is the world's fifth largest IT solution company and the second biggest in Finland. The company offers consulting, systems integration, outsourcing of IT and business processes and implementation of IT solutions.

In this thesis, a monitoring program was planned and implemented. The program can create monitoring routines, which are executed in a customer environment. The purpose of the monitoring program was to make monitoring faster, to collect monitoring information and process it to historical data.

Key words: Data Warehouse, Microsoft Azure Cloud Service, .NET Framework, Windows Communication Foundation, Entity Framework, monitoring

SISÄLLYS

1 JOHDANTO

Tiedon kerääminen on liiketoiminnan kannattavuuden seuraamisen vuoksi tärkeää yrityksille. Yritysten keräämä tieto liittyy niin laskutukseen, henkilöstöön, myyntiin ja moneen muuhun liiketoimintaan vaikuttavaan osa-alueeseen. Tietoja kerätään eri lähteistä ja ne yleensä pyritään yhdenmukaistamaan ja tuomaan yhteen hallittuun tietovarastoon, josta tieto voidaan jalostaa erillisille raporteille ja analyysijärjestelmiin. Näiden tietojen pohjalta voidaan analysoida ja suunnitella liiketoimintaa eteenpäin.

Tiedonkeruussa ei ole tärkeää pelkästään suurien tietomäärien käsittely, vaan tärkeää on myös tiedon oikeellisuus. Tehtäessä liiketoimintaan liittyviä päätöksiä tulee olla varma, että käytetty tieto on ajantasalla eikä sisällä virheitä. Tiedonkeruujärjestelmät ovat monesti moniosaisia ja sisältävät paljon eri lataajia eri lähteille, koska lähteinä voivat toimia niin Excel-tiedostot kuin toiset tietovarastot. Tällöin on tärkeää seurata, että eri tietolataajat menevät varmasti onnistuneesti läpi. Tietolataajien oikeanmukainen toiminnallisuuden vaatii tietovarastoa ja sen lataajia suunnitellessa myös panostusta lokitietojen tallentamiseen. Tällöin voidaan valvoa latausajojen läpimenoa ja kestoja.

Tietolataajien ja tiedon oikeellisuutta seurataan erilaisilla valvontakäytännöillä. Valvonnat voidaan tehdä automatisoidusti skripteillä tai ohjelmilla, mutta ne voidaan myös suorittaa käsin, ja niistä vastaavat yleensä yrityksen tietovarastoa hallinnoivat tahot. Asiakkaana olevat yritykset haluavat kuitenkin saada valvonnat suoritettua tehokkaasti ja mahdollisimman aikaisin aamusta, jotta päivän aikana tehtävät liiketoimintapäätökset perustuisivat ajantasalla olevaan tietoon. Mikäli valvonnoissa havaitaan virheitä, niin niihin on tärkeää reagoida nopeasti ja saada tieto ajantasalle. On tärkeää, että asiakas tietää mahdollisimman nopeasti tiedon olevan väärää tai vanhentunutta, jotta sen johdosta ei tehtäisi päätöksiä.

Tässä opinnäytetyössä tutkimusongelmana on automatisoidun valvonnan toteuttaminen asiakasympäristöjen päivittäisvalvontaan. Tämä

opinnäytetyö keskittyy automatisoidun valvonnan valvontarutiinien luomiseen siihen tarkoitukseen toteutetulla C#-sovelluksella. Sovellus hoitaa CGI Suomi Oy:n asiakkaiden ympäristöjen valvontojen valvontarutiinien luonnit, jotka viedään asiakkaan ympäristöön. Sovellus on suunniteltu Microsoft Windows -ympäristöihin ja Microsoft-työkaluilla tehdyille tiedonkeruuratkaisuille Ympäristössä oleva ohjelma ajaa luodut rutiinit ja ilmoittaa, mikäli löytyy poikkeamia. Tämän jälkeen kerätty tieto viedään Azure-tietovarastoon, josta se tuodaan raporteille nähtäväksi, jolloin poikkeamatilanteiden löytyminen nopeutuu. Valvontasovelluksen tarkoituksena on nopeuttaa ja yksinkertaistaa aamuvalvontojen tekoja. Lisäksi sovelluksen tarkoituksena on kerätä historiatietoa tulleista virheistä, jolloin toistuviin virheisiin voidaan puuttua tehokkaamin.

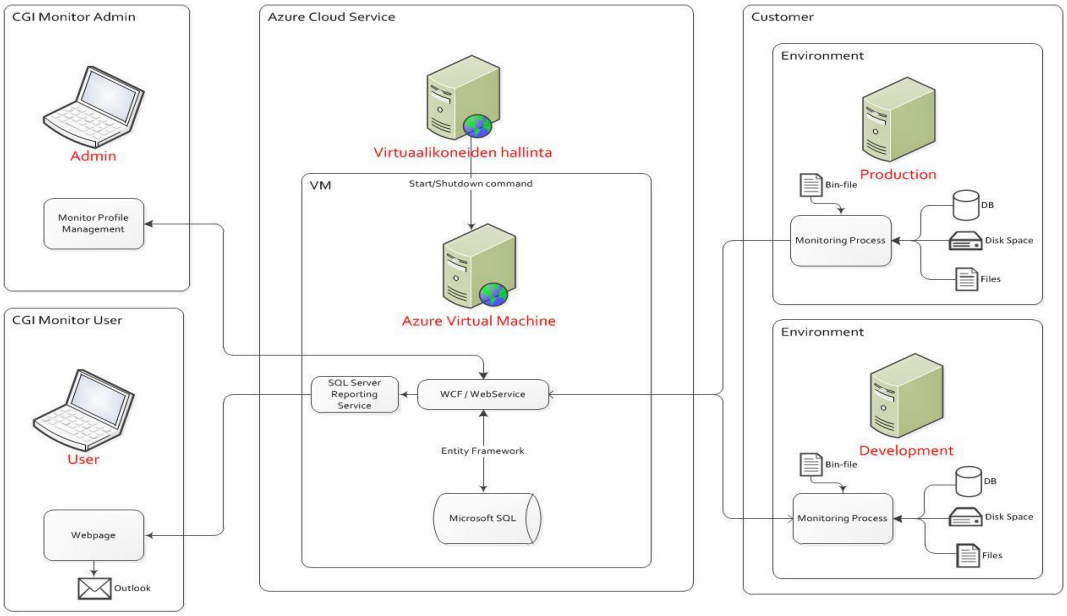
2 TOIMINTAYMPÄRISTÖKUVAUS

Toimintaympäristö koostuu kolmesta osasta:

- CGI Suomi Oy:n ympäristöistä
- Microsoft Azure -palvelimesta
- asiakkaiden ympäristöistä.

CGI:n ympäristössä luodaan valvontasovelluksella rutiinit, jotka tallennetaan Azure-palvelimella olevalle virtuaalikoneelle tietokantaan. Tämän lisäksi valvontasovellus luo erillisen bin-tiedoston, joka sisältää asiakasympäristön valvonnat, ja tämä bin-tiedosto viedään manuaalisesti asiakasympäristöön, jossa valvonnat tehdään. Valvontaprosessi ajaa kyseisen tiedoston avulla valvonnat läpi asiakasympäristössä ja palauttaa valvontatulokset Azure-palvelimelle tietokantaan. Tämän jälkeen valvontatuloksia voidaan tarkastella CGI:n ympäristössä SSRS-raportteilta (SQL Server Reporting Service), jotka luodaan tietokannasta.

Tiedonsiirto Azure-palvelimen kautta on toteutettu WCF-yhteydellä, joka on yhteydessä Entity Frameworkin avulla samalla palvelimella olevaan tietokantaan. Kuvio 1 kuvastaa yhteyksiä ympäristöjen välillä.



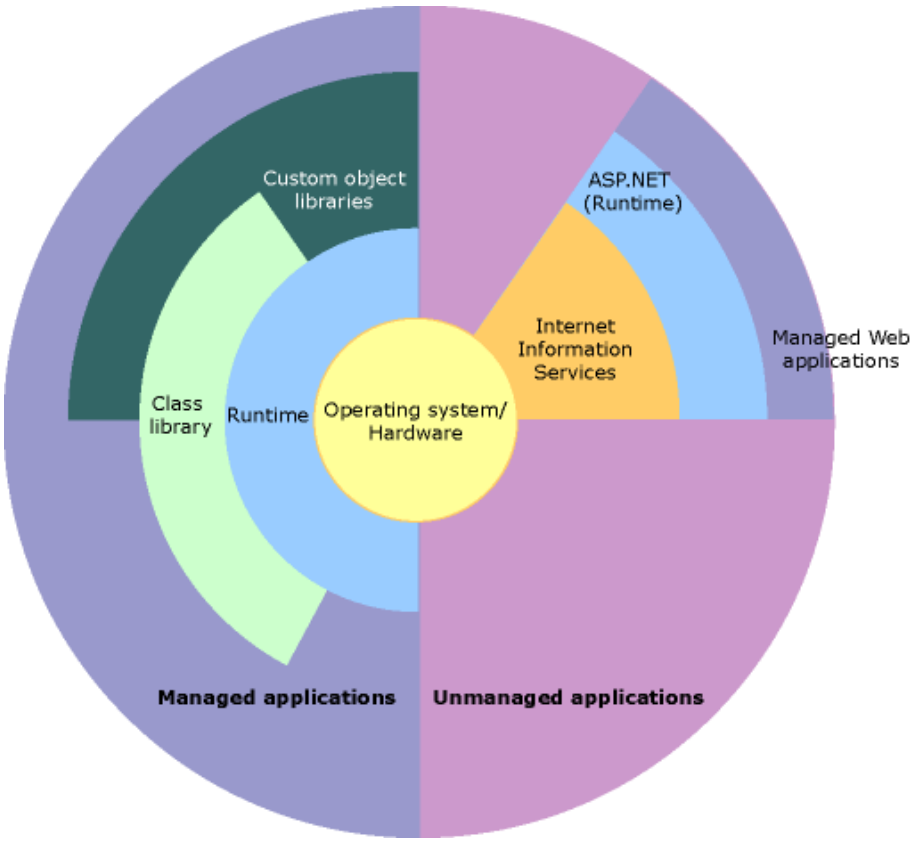
KUVIO 1. Ympäristökuvaus

3 .NET FRAMEWORK

.NET on yksi tehokkaimmista Microsoftin julkaisemista tekniikoista. Sen tarkoituksena on täydentää ajatus ohjelmistosta kuin se olisi palvelu. .NET-rajapinta auttaa kehittäjiä raketamaan Windows- ja web-sovelluksia, kuten myös web-palveluita, hyödyntäen eri ohjelmistokieliä esimerkiksi C# ja Visual Basic. (Kogent Solutions Inc. 2008, 1.)

Microsoft julkitsi .NET-aloitteen kesäkuussa 2000. .NET-alusta on kehitysrajapinta, jossa on ohjelmoitavia rajapintoja Windows-palveluille ja -ohjelmointirajapinnoille (API). .NET integroi monia teknologiaoita, joita Microsoft kehitti 1990-luvun lopulla. (Thai & Lam 2002, 7.)

.NET rajapinnan ensimmäinen versio julkaistiin 13. helmikuuta 2002. Tämä versio rajapinnasta on myös ensimmäinen julkaisu Microsoft Visual Studio .NET:lle, joka tunnetaan Visual Studio .NET 2002 -nimellä. (Kogent Solutions Inc. 2008, 1.) Kuvio 2 näyttää .NET Frameworkin rakenteen.



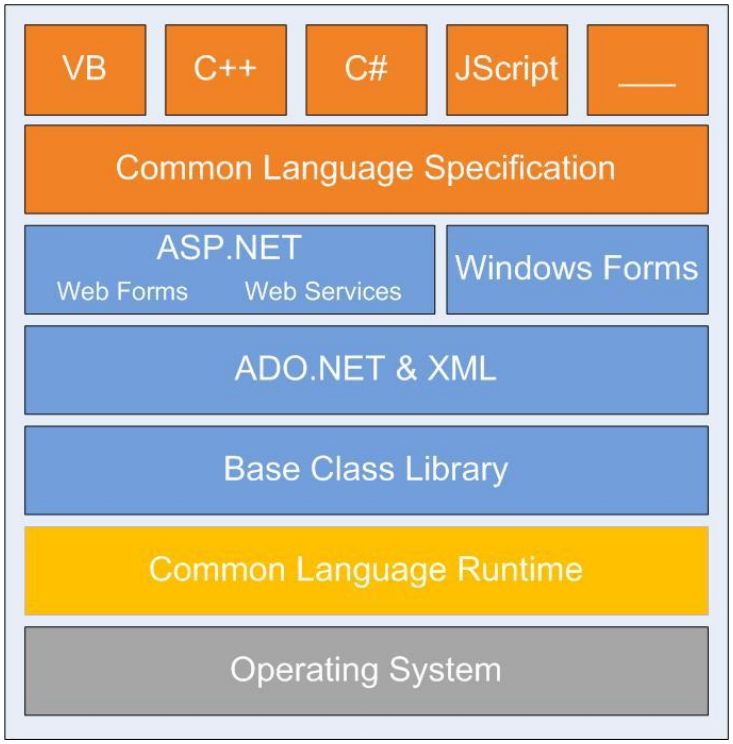
KUVIO 2. .NET Framework (Overview of the .NET Framework 2015)

3.1 Microsoft .NET -arkkitehtuuri

.NET-rajapinta on sisäänrakennettu Windows-komponentti, joka tukee ohjelmien ja web-palveluiden muodostamisen (building) ja suorittamisen. .NET-rajapinta on kehitetty ottamaan huomioon seuraavat tavoitteet ja päämäärät:

- tarjoamaan johdonmukaisen olio-ohjelmointi ympäristön, jossa käyttäjä voi kehittää sovelluksia
- tarjoamaan lähdekoodin ajamiseen ympäristön, joka takaa koodin turvallisuuden sitä ajaessa
- tarjoamaan lähdekoodin ajamiseen ympäristön, joka yksinkertaistaa käyttöönoton (deployment) ja versioinnin
- tarjoamaan koodin ajamiseen ympäristön, joka eliminoi ongelmat käyttäessä komentosarjaympäristöä huomioiden sen suorituskyvyn
- tarjoamaan perinteisen ohjelmointimallin, jossa ohjelmointikielen valinta on käyttäjän käsissä.

.NET-rajapinta koostuu kahdesta komponentista: Common Language Runtimesta (CLR) ja .NET Framework Class Librarysta (FCL), jota yleisemmin nimitetään Base Class Libraryksi. CLR on perusta .NET-rajapinnalle, ja se tarjoaa monia palveluita ohjelmistojen käytettäväksi. Se myös muodostaa ympäristön, jota muut ohjelmistot käyttävät. Luokkakirjasto (FCL) on olio-ohjelmointipohjainen kokoelma uudelleenkäytettäviä luokkia, jotka tarjotaan kaikille palveluille ja tietorakenteille, joita ohjelma voisi tarvita. Kuvio 3 näyttää .NET-rajapinnan hierarkiarakenteen ja siihen kuuluvat työkalut. Se myös esittää CLR- ja FCL-komponenttien suhteen muihin ohjelmistoihin ja järjestelmän kokonaisuuteen. (Kogent Solutions Inc. 2008, 3.)



KUVIO 3. .NET-arkkitehtuuri

Kuviosta 3 voi huomata, että käyttöjärjestelmä on pohjana rakenteelle. Se voi olla mikä tahansa Microsoft Windows -käyttöjärjestelmä. Tulee kuitenkin huomioida kehittäessä sovelluksia .NET-rajapinnoilla, niin uusimmat versiot rajapinnasta eivät ole enää yhteensopivia vanhojen Windows-käyttöjärjestelmien kanssa.

Käyttöjärjestelmäkerroksen päälle rakentuu CLR-kerros, joka on alustana lähdekoodille ja erottaa sen käyttöjärjestelmästä. CLR tarjoaa palvelut ohjelmistoille ja tarjoaa myös perusjoukon kirjastoluokkia (Base Class Library). Ohjelmistokielikääntäjien, jotka luovat koodin CLR:lle, täytyy sitoutua perusjoukkoon määrittelyitä, jotka on vakiinnutettu Common Language Specification -tasolla (CLS). Tämän kerroksen yläpuolelta löytyvät kaikki .NET-ohjelmointikielet. (Kogent Solutions Inc. 2008, 4.)

3.2 Common Language Runtime

Common Language Runtime (CLR) on alusta, jossa ohjelmistot ovat hallinnoitu ja suoritettu. CLR:ä käytetään hallitsemaan muistia, säikeiden

ja lähdekoodin suorittamista, lähdekoodin turvallisuutta, verifointia, kääntämistä ja muita järjestelmän palveluita. Koska CLR hallinnoi lähdekoodin ajamista, niin lähdekoodia, joka toimii CLR kanssa, sanotaan hallituksi koodiksi (managed code). CLR myös tukee palveluita, joita ohjelmistot voivat käyttää päästäkseen hyödyntämään monia resursseja, esimerkiksi taulukoita ja käyttöjärjestelmän kansiota.

CLR:n hyötyjä on hallittu ajoympäristö, jonka avulla koodikääntäjät ja työkalut paljastavat CLR:n toiminnallisuuden ja helpottavat lähdekoodin luontia. Yhteentoimivuus hallitun ja ei-hallitun koodin välillä mahdollistaa kehittäjien jatkamaan välttämättömien COM-komponenttien ja DDL:ien käyttöä. Hallittu ajoympäristö poistaa monia tavallisimpia ohjelmisto-ongelmia, kuten esimerkiksi objektien muistivarauksien vapauttaminen, kun niitä ei enää ajon aikana tarvita. Tämä automaattinen muistinhallinta ratkaisee muistivuotoja ja epäkelpoisia muistiviitteitä. (Kogent Solutions Inc. 2008, 4.)

3.3 .NET-ohjelmointi

Ilman .NET-rajapintaa ohjelmoijat joutuvat valitsemaan ohjelmointirajapinnan ja kirjastot, jotka tukevat järjestelmän palveluita. Valittuaan käytettävän kirjaston, ohjelmoijan on opittava käyttämään rakenteita, luokkia, funktioita ja rajapintoja, joita kirjasto tarjoaa. Nämä ovat myös riippuvaisia järjestelmäympäristöstä, eivätkä ne siirry suoraan ympäristöstä toiseen.

Yksi .NET-rajapinnan tavoitteista on tuoda yhdenmukaisuutta ohjelmien kehitykseen tarjoamalla kehittäjille viitekehys yleisten luokkien käyttämiseksi. Yleinen viitekehys on käytännöllinen, jos kehittäjä tietää kuinka hyödyntää siirräntä-toiminnallisuutta yhdellä koodikielellä .NET-käyttöympäristössä voi helposti hyödyntää samaa myös toisella .NET-koodikielellä. Tämä on mahdollista nimiavaruuksien, luokkien, metodien ja muiden ominaisuuksien avulla, joilla on johdonmukainen esitysmuoto kaikissa koodikielissä.

Jokainen tyyppi .NET:ssä on objekti, joka tarkoittaa sitä, että se tulee johtaa suorasti tai epäsuorasti objekti-luokasta. Objekti-luokka tukee samankaltaisuutta, jonka jokainen .NET-luokka perii ja automaattisesti tarjoaa sen luokkaa käyttäville. Objekti-luokka tarjoaa julkisia metodeita (taulukko 1), joita voidaan kutsua missä tahansa .NET-objektissa ajon aikana. (Thai & Lam 2002, 42.)

TAULUKKO 1. Objekti-luokan julkiset metodit

Metodi	**Kuvaus**
Equals()	Vertaa kahta objektia ja määrittää ovatko samat (sama sisältö).
ReferenceEquals()	Vertaa kahden objektin referenssiä ja määrittää referoivatko ne samaan objektiin.
GetHashCode()	Palauttaa objektin tiivistekoodin. Tiivistekoodia käytetään .NET:ssä lisäämään mekanismi määrittämään objektin uniikillisuus ajon aikana.
Get Type()	Palauttaa objektin tyypin ajon aikana. Objektin tyypin avulla voidaan hyödyntää kaikkia objektin tarjoamia ominaisuuksia.
ToString()	Palauttaa merkkijono-esitysmuodon objektista.

3.4 Windows Form

Windows Form tarjoaa yhtenäisen ohjelmointimallin stantardin Windows-ohjelman kehittämiseen. Se on samankaltainen kuin natiivi Windows-ohjelmistirajapinta huomioiden abstraktiotason, mutta se on paljon monipuolisempi ja tehokkaampi. Sen ollessa osana .NET-rajapintaa, se mahdollistaa integraation esimerkiksi Web-palveluiden, ADO.NET:n ja .NET-luokkien kanssa.

Windows Formin arkkitehtuuri on melko yksinkertainen. Se koostuu kontrollereista ja säiliöistä (containers). Monet käyttöliittymän luokista tulevat kontrolleriluokasta. Voidaan sanoa, että kaikki objektit, jotka sisältyvät Windows Formin, ovat kontrollereita. Mikäli kontrolleri sisältää toisen kontrollerin, niin tällöin kyseessä on säiliö. Ohjelman käyttöliittymä toimii pääsäiliönä muille kontrollereille ja säiliöille. (Thai & Lam 2002, 204 – 205.)

3.4.1 Kontrolleriluokka

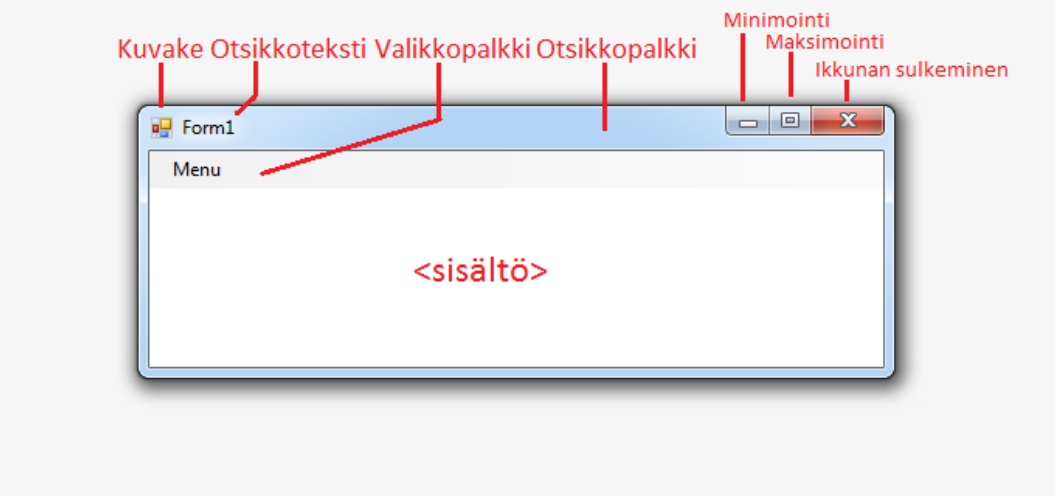
Kontrolleri on pohjaluokka kaikille käyttöliittymän kontrollereille Windows Form -ohjelmassa. Se tarjoaa perusominaisuudet kaikille kontrollereille esimerkiksi käyttäjäsyötteen hallintaan. Taulukossa 2 on listattu osa kontrolleri-luokan tarjoamista ominaisuuksista, metodeista ja tapahtumasta. (Thai & Lam 2002, 206.)

TAULUKKO 2. Kontrollerin ominaisuuksia, metodeita ja tapahtumia

Ominaisuudet	**Selitys**
Controls	Nämä ominaisuudet mahdollistavat rakentamaan kontrollereille hierarkian. Controls-ominaisuus listaa kaikki lapsikontrollerit, kun Parent-ominaisuus osoittaa kantakontrolleriin.
Parent	
Enabled	Nämä ominaisuudet määrittelevät kontrollerin visuaalista tilaa.
Focused	
Visble	
Top	Nämä ominaisuuet määrittelevät kontrollerin sijaintia ja kokoa.
Left	
Right	
Bottom	
Width	
Height	
Size	
Metodit	**Selitys**
Show	Nämä metodit manipuloivat kontrollerin visuaalista tilaa
Hide	
Focused	
Select	
Refresh	Nämä metodit kontrolloivat milloin ja mitä näytönosioita tulee uudelleenpiirtää. Refresh-metodi pakottaa kontrollerin välittömästi piirtämään itsensä ja sen lapsikontrollerit uudelleen.
Invalidate	
Update	
Tapahtumat	**Selitys**
Click	Nämä tapahtuvat määrittelevät kontrollerin käyttäytymistä esimerkiksi hiirellä painaessa.
MouseDown	
MouseUp	
MouseMove	
MouseWheel	
KeyDown	Samankaltaisia metodeita kuin hiirellä toimiessa, mutta ovat vain näppäimistölle.
KeyUp	
KeyPress	

3.4.2 Lomakeluokka

Lomake Windows Formissa on samankaltainen konsepti kuin sivu on Web-lomakkeessa. Se on säiliötyyppinen kontrolleri, joka pitää sisällään kaikki käyttöliittymän kontrollerit. Lomake-objektin ominaisuuksia voidaan manipuloida hallitakseen lomakkeen ulkoasua, kokoa ja värejä. Vakiomuotoinen lomake sisältää otsikkopalkin, joka sisältää kuvakkeen, otsikkotekstin ja kontrollerilaatikon lomakkeen koon minimointiin, maksimointiin ja lomakkeen sulkemiseen. Lomakkeikkunan rajat rajoittavat sisällön näkyvyyttä, ja niiden avulla voidaan suurentaa ja pienentää lomakeikkunaa. Lomakkeeseen voidaan lisätä myös valikkopalkki, joka tulee otsikkopalkin alapuolelle, ja vierityspalkki, jonka avulla voidaan saada lomakkeeseen enemmän sisältöä, mutta nämä eivät ole välttämättömiä. Kuvio 4 kuvastaa lomakeikkunan rakennetta. (Thai & Lam 2002, 207.)



KUVIO 4. Tyhjä lomakeikkuna

3.4.3 Ohjelmaluokka

Ohjelma-luokka tarjoaa staattiset metodit käynnistää ja lopettaa ohjelma, tai suodattaa Windows-viestejä ohjelmassa. Kaikki Windows Form -

ohjelmat sisältävät referenssin tähän ohjelmaluokkaan. Ohjelmaluokan tarjoaman run-metodin avulla käynnistetään lomake.

Ohjelma-luokka tarjoaa myös muita metodeja ja ominaisuuksia, mutta run-metodi on tärkein näistä kaikista. Run-metodi käynnistää ohjelman säikeen viestisilmukan. Tällä metodilla on kaksi signeerausta. Kuviossa 5 on signeeraus, joka ei sisällä parametreja, ja sitä normaalisti käytetään ei-graafisen käyttöliittymän ohjelmissa.

```
System.Windows.Forms.Application.Run(  );
```

KUVIO 5. Run-metodi ilman parametreja

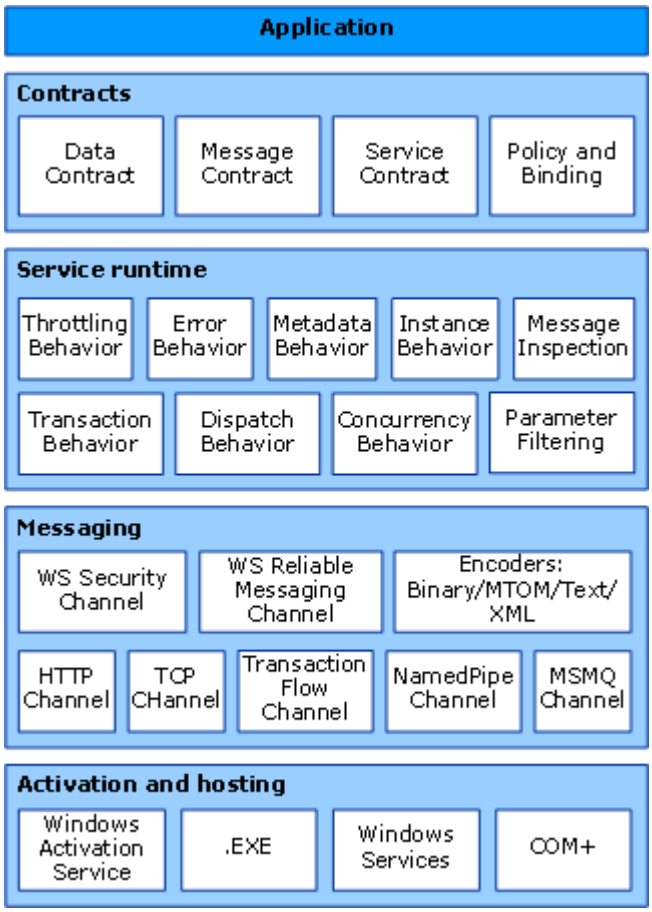
Kuviossa 6 on toinen signeeraus, joka ottaa lomakkeen parametrina. Lomake ’MyForm’ on tulokohta graafisen käyttöliittymän Windows Form -ohjelmalle. (Thai & Lam 2002, 208.)

```
System.Windows.Forms.Application.Run(new MyForm());
```

KUVIO 6. Run-metodi parametrilla

4 WINDOWS COMMUNICATION FOUNDATION

Windows Communication Foundation (WCF) on Microsoftin kehittämä yhtenäinen ohjelmointimalli, jonka avulla voidaan kehittää palveluorientoituneita ohjelmia. Se mahdollistaa kehittäjien luoda turvallisia, luotettavia ja siirrettäviä ratkaisuja, jotka integroituvat eri alustoilla. WCF on rakennettu .NET-rajapintaan, ja se yksinkertaistaa yhdistettyjen järjestelmien kehitystä. (Liu 2012, 14.) Kuviosta 7 on nähtävissä WCF:n arkkitehtuuri.



KUVIO 7. WCF-arkkitehtuuri (Windows Communication Foundation Architecture 2015)

4.1 SOA

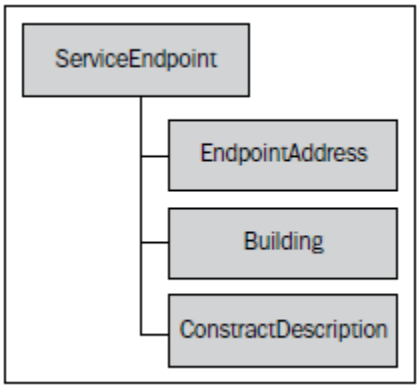
SOA tarkoittaa palveluorientoitunutta arkkitehtuuria (service-oriented architecture). SOA on arkkitehtuurinen suunnittelumalli, jossa useat pääperiaatteet määrittelevät suunnitellun luonteen. Pohjimmiltaan SOA

määrittelee, että jokaisen komponentin järjestelmässä tulisi olla palvelu ja järjestelmän tulisi koostua useasta riippumattomasta palvelusta. Palvelu tässä yhteydessä tarkoittaa ohjelman osaa, joka palvelee prosessia. Riippumattomalla tarkoitetaan, että näiden palveluiden tulisi olla itsenäisiä toisistaan, joten yhden vaihtaminen ei tulisi vaikuttaa muihin palveluihin.

SOA ei ole toteutusteknologia tai ohjelmointikieli, vaan järjestelmäsuunnitelman lähestymistapa. Se on arkkitehtuurinen malli, jonka tavoitteena on parantaa tehokkuutta, järjestelmän ketteryyttä ja lisätä tuottavuutta järjestelmään. Keskeiset käsitteet ovat palvelut, korkea yhteentoimivuus ja palveluiden keskinäinen vähäinen riippuvuus. (Liu 2012, 8.)

4.2 Päätepisteet

Viestit lähetetään asiakassovelluksen ja palvelun päätepisteiden välillä. Päätepisteet ovat paikkoja, joista viestit lähetään, vastaanotetaan tai molempia. Niissä määritellään kaikki tarvittava tieto viestien käsittelyyn. WCF-palvelun päätepisteet koostuvat osoitteesta, sidoksesta ja palvelusopimuksesta (kuvio 8). On myös tärkeää, että päätepisteet ovat määritelty vastaamaan toisiaan.(Liu 2012, 22.)



KUVIO 8. Päätepisteen rakenne (Liu 2012, 23)

4.3 Osoitteet

Palvelun osoitteellistaminen on oleellinen osa WCF:n käyttöä. Palvelulla täytyy olla osoite, jotta siihen voidaan lähettää viestejä. Osoitteet WCF:ssä ovat URL-muodossa, joissa määritellään käytettävä protokolla, palvelua ylläpitävä laite ja palvelun osoitepolku. Porttinumero ei ole välttämätön, ja sen käyttö riippuu käytettävästä protokollasta. Taulukossa 3 on eriteltynä WCF:n osoiterakenne (Pathak 2011, 58.)

TAULUKKO 3. WCF:n osoiterakenne (Pathak 2011, 58)

Osoitteen osio	**Tarkempi kuvaus**
Siirtoprotokolla	Määrittelee siirtoprotokollan tai -skeeman
Laitteen nimi	Määritteleen laitteen palvelin nimen
Portti	Tämä on vaihtoehtoinen kenttä ja määritellään : portti
Polku	Tämä on palvelun tiedostopolku.

Palvelun osoite siis koostuu seuraavasti:
siirtoprotokolla://<laitteen nimi>[:portti]/polku1/polku2

4.4 Sidokset

Sidokset ovat koottuja valintoja käytettävästä siirtoprotokollasta, viestin koodauksesta, luotettavuudesta, turvallisuudesta, siirron etenemisestä ja yhteentoimivuudesta. WCF hyödyntää sidoksia päätepisteen yhdistäessä toiseen päätepisteeseen tai kommunikoidessa toisen päätepisteen kanssa. Kommunikoidessa päätepisteiden kanssa sidoksissa voidaan määritellä, onko viestintä synkronista, asynkronista vai dublex-mallista. Yksittäinen palvelu voi tukea useampaa sidosta eri osoitteille. Taulukosta 4 käy ilmi WCF:n viisi yleisintä sidosta. (Löwy 2010, 24.)

TAULUKKO 4. WCF:n viisi yleisintä sidosta (Löwy 2010, 25)

BasicHttpBinding	Oletuksena käytettävä sidos, jossa on HTTP-protokolla. Tämän avulla voidaan saada vanhatkin verkkopalvelut toimimaan uusien palveluiden kanssa
NetTcpBinding	TCP-sidos käyttää TCP:tä laitteiden väliseen kommunikointiin. Vaatii sekä asiakasohjelman, että palvelun käyttävän WCF:ää.
NetNamedPipeBinding	IPC-sidos hyödyntää nimettyjä putkia tiedonsiirrossa saman laitteen sisällä olevien palveluiden välillä. Se on turvallisin sidos, koska se ei hyväksy pyyntöjä laitteen ulkopuolelta.
WSHttpBinding	WS-sidos (Web Services) hyödyntää HTTP:tä tai HTTPS:ää tiedonsiirrossa ja tarjoaa turvallisen siirron internetin yli.
NetMsmqBinding	MSMQ-sidos (Microsoft Message Queuing) hyödyntää MSMQ:ta tiedonsiirrossa ja tarjoaa tuen epäonnistuneille jonotuskutsuille.

4.5 Sopimukset

Sopimukset asettavat ehdot, jotka määrittävät WCF:n rajapinnan. WCF-palvelu kommunikoi muiden ohjelmien kanssa sen sopimukset asettamissa rajoissa. On olemassa monen tyyppisiä sopimuksia, kuten palvelusopimus (service contract), operaatiosopimus (operation contract), tietotyyppisopimus (data contract) viestisopimus (message contract) ja virhesopimus (fault contract).

Tärkein sopimuksista on palvelusopimus, joka toimii rajapintana WCF-palvelulle. Se kuvastaa muille, mitä palvelu voi suorittaa. Se voi sisältää palvelutasoisia asetuksia, kuten palvelun nimen, nimiavaruuden ja oikean tavan kutsua palvelun muita sopimuksia. WCF-palvelun on sisällettävä vähintää yksi palvelusopimus palvelupyyntöjä varten.

Operaatiosopimukset määritellään palvelusopimuksen kanssa. Ne määrittävät käytettävät parametrit ja palautusarvot käytettävissä olevissa operaatioissa. Mikäli operaatiosopimuksen täytyy toimittaa viesti tai palauttaa palautusarvona viesti, niin tämäntyyppiset viestit määritellään viestisopimuksella. Tietotyyppisopimukset kertovat mitkä tietotyypit ovat käytössä WCF-palvelussa. Kaikkien käytettävien tietotyypien pitää olla kuvattuna metatiedoissa. Virheiden tilanteiden hallinta operatiivissa sopimuksissa on määritelty virhesopimuksessa. Sen mukaan määritellään, kuinka monta virhettä palvelu voi käsitellä keskeyttämättä toimintaansa. (Liu 2012, 18.)

Kuviossa 9 on WCF-palvelun tarjoama rajapinta MyService. Se esitellään [ServiceContract]-attribuutilla ja sen toiminnot [OperationContract]-attribuuteilla. PuchaceOrder on [DataContract]-attribuutilla määritelty oma luokka, jonka tietotyyppien aksessorit määrittää [DataMember]-attribuutti.

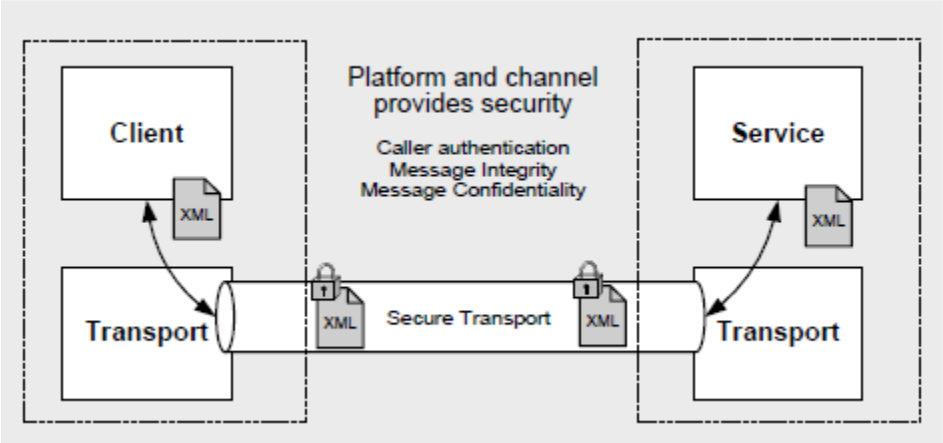
```
[ServiceContract]
interface MyService
{
    [OperationContract]
    [FaultContract(typeof(FaultException))]
    double SquareRoot(int root);

    [OperationContract]
    [FaultContract(typeof(FaultException))]
    double Sum(int num1, int num2);

}

[DataContract]
public class PurchaseOrder
{
    private int poId_value;

    [DataMember]
    public int PurchaseOrderId
    {
        get { return poId_value; }
        set { poId_value = value; }
    }
}
```

KUVIO 9. MyService-rajapinta

4.6 Tietoturva

Minkä tahansa palvelukeskeisen arkkitehtuurin tulee tukea turvallisuusominai-suuksia, jotka tarjoavat auditoinnin, autentikoinnin, auktorisoinnin, tietosuojan ja tiedon eheyden viestien vaihtoon asiakasohjelman ja palvelun välillä. WCF tarjoaa nämä kaikki turvallisuusominaisuudet, ja ne sisältyvät WCF:n kolmeen ominaisuuteen: tiedonsiirron turvallisuuteen, auktorisointiin ja auditointiin. WCF tarjoaa pääsyn näihin ominaisuuksiin sidosten ja käyttäytymisreferenssin kautta. Sidosta valittaessa tulee huomioida eri sidosten turvallisuustasot, jotka jakautuvat tietoliikennesalaukseen ja viestien salaukseen.

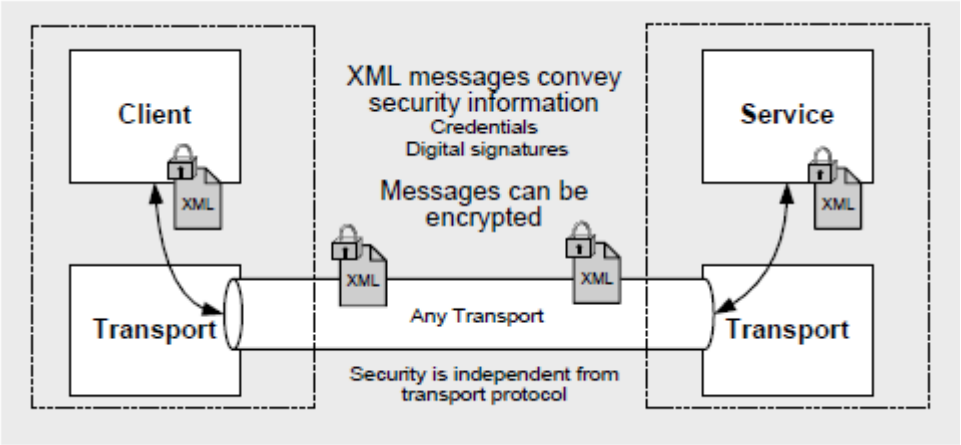
Tietoliikennesalausta käytettäessä koko käytettävä kommunikaatiokanava salataan (esimerkiksi käyttämällä SSL:ä) kahden päätepisteen välille (point-to-point salaus). Päätepisteen halutessa lähettää toiselle päätepisteelle tietoa tulee tällöin luoda uusi salattu yhteys pisteiden välille. Kuvio 10 kuvastaa tietoliikennesaluksen toimintaa. (Microsoft 2008, 88 – 89.)



KUVIO 10. Tietoliikennesalaus (Microsoft 2008, 90)

Viestien salaamista käyttäessä jokainen viesti suojataan erikseen. Tämä antaa enemmän joustavuutta autentikoinnin kannalta, ja viestiä lähetettäessä voidaan käyttää mitä tahansa tiedonsiirtoprotokollaa,

kunhan vain asiakasohjelma ja palvelu tukevat sitä. Kuvio 11 kuvastaa viestien salaukseen perustuvaa tekniikkaa. (Microsoft 2008, 90.)

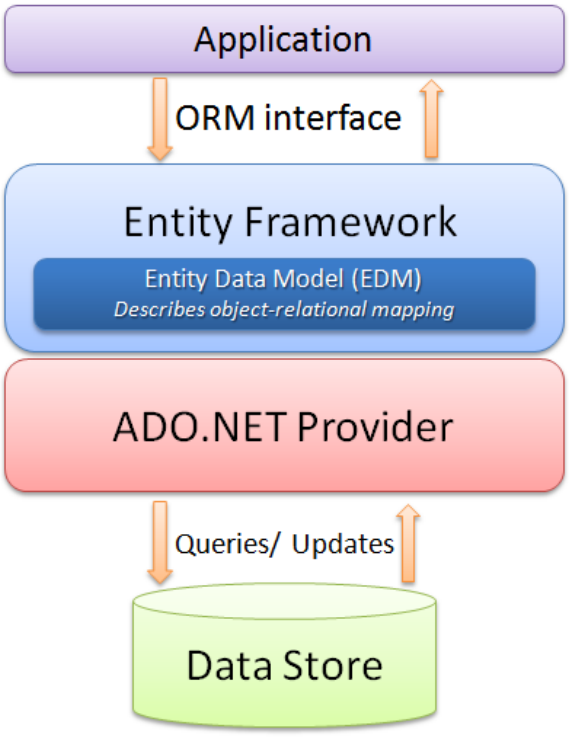


KUVIO 11. Viestien salaus (Microsoft 2008, 91)

5 ENTITY FRAMEWORK

Entity Framework julkaistiin ensimmäisen kerran kesällä 2008 osana Visual Studio 2008 Service Pack 1:tä ja .NET 3.5 Service Pack 1:tä. Sen tarkoituksena oli auttaa kehittäjiä käyttämään tietokantojen tauluja, näkymiä ja tallennettuja toimintasarjoja (stored procedures) tietämättä niin paljoa näiden nimistä ja tietokannan skeemasta. (Lerman 2010, 2.)

Entity Framework on osa Microsoftin ADO.NET-rajapintaa, joka tarjoaa tietolähteisiin käsittelyyn tarvittavia palveluita .NET-rajapinnassa. Entity Framework on Object/Relational Mapping (ORM) -rajapinta, joka vähentää yhteensopivuusongelmia .NET-rajapinnan olio-ohjelmoinnin ja relaatiotietokantojen välillä. Kuvio 12 kuvastaa Entity Frameworkin arkkitehtuuria. (ADO.NET Entity Framework At-a-Glance 2015.)



KUVIO 12. Entity Frameworkin arkkitehtuuri (ADO.NET Entity Framework At-a-Glance 2015)

5.1 Object-Relational Mapping

Object-Relational Mapping on ohjelmointitekniikka, joka sisältää joukon luokkia, jotka kartoittavat relaatiotietokannan entiteetin objektiksi. Alkuaan ohjelmat ovat voineet kutsua ennaltamääritetyn natiivin tietokannan ohjelmointirajapintaa keskustellakseen tietokannan kanssa. Tätä varten kehitettiin ODBC (Open Database Connectivity) yhdistämään kaikki kommunikaatioprotokollat useanlaisiin relaatiotietokantojen hallintajärjestelmiin eli RDBMS:n (relational database management systems). ODBC suunniteltiin olemaan itsenäinen ohjelmointikielien, tietokantajärjestelmien ja käyttöjärjestelmien suhteen. ODBC:n avulla yksi sovellus voi kommunikoida useamman eri RDBMS:n kanssa käyttäen samaa lähdekoodia, mutta korvaamalla vain perustana olevat ODBC-ajurit.

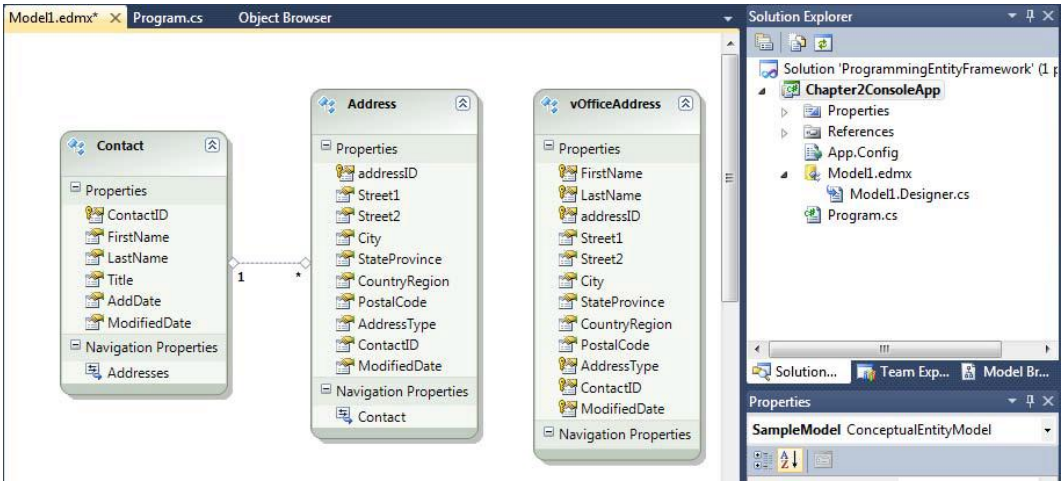
Välittämättä siitä, mitä metodia käytetään yhdistäessä tietokantaan, niin tietokannasta tuleva data on esitettävä tietyssä muodossa sovelluksessa. Esimerkiksi, jos Tilaus-taulun rivi palautetaan tietokannasta, on oltava muuttuja tilausnumerolle ja joukko muuttujia tilausrivin mahdollisille lisätiedoille, jotka saattavat tulla toisesta taulustakin, joka on yhdistetty viiteavaimella Tilaus-tauluun. Vaihtoehtoisesti sovellukseen voidaan luoda luokka tilauksille ja toinen luokka näille mahdollisille lisätiedoille. Ja kun luodaan toinen sovellus, niin tämä joudutaan tekemään uudelleen tähän toiseenkin sovellukseen. Tällaisessa tilanteessa ORM on hyödyllinen apu.

ORM:n avulla jokainen tietokanta on esitetty ORM-viitekehysobjektina tietyllä ohjelmointikielellä ja tietokantaentiteeteillä, kuten taulut esitetään luokkina ja taulujen linkitykset näiden luokkien välisenä yhteytenä. Esimerkiksi ORM luo Tilaus-luokan kuvastamaan Tilaus-taulua ja TilausLisatiedot luokan kuvastamaan Tilaus Lisatiedot -luokkaa. ORM on vastuussa kartoituksista ja yhteyksistä näiden luokkien ja tietokannan välillä. Tietokanta on tällöin täysin kuvauttuna näillä luokilla sovelluksessa. Sovelluksen täytyy vain käsitellä näitä luokkia fyysisen tietokannan sijasta. Sovelluksen ei tarvitse tällöin enää huolehtia yhteydestä tietokantaan, SQL-kyselyiden muodostamisesta, kyselyiden samanaikaisuuden

toimivuuden varmistamisesta tai siitä, kuinka yleisiä transaktioita käsitellään. ORM hallinnoi kaikki tietokantaan liittyvät toimet. (Liu 2012, 168.)

5.2 Entity Data Model

Entity Data Model (EDM) on silta ohjelman ja tietovaraston välillä. EDM tarjoaa mahdollisuuden työskennellä tietovaraston tiedon kanssa ottamatta suoraan yhteyttä tietovarastoon. .NET-ohjelmistorajapinta tarjoaa EDM:n avulla integraation tietovarastoon, jonka avulla voidaan hakea tai tallettaa tietoa. Entity Framework luo tästä mallista luokat, joita voidaan hyödyntää suoraan lähdekoodissa. Kuvio 13 näyttää tyypillisen taulujoukon skeeman, joka on luotu tietokannasta.



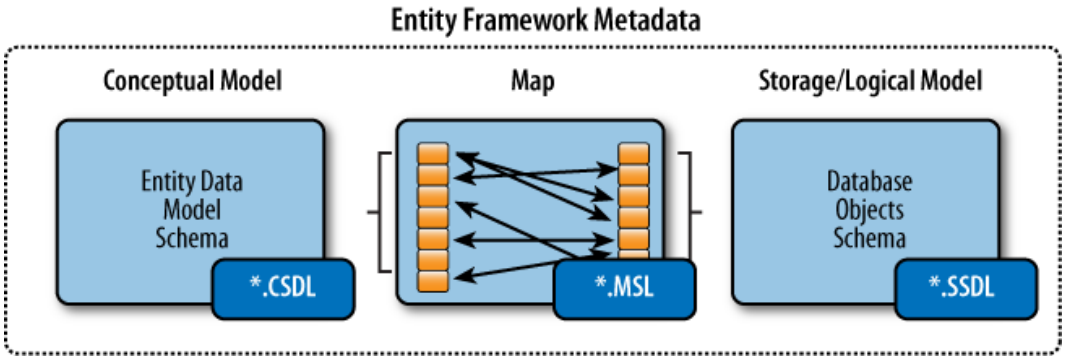
KUVIO 13. Model1.edmx kuvastaa tietokannassa olevat taulut (Lerman 2010, 20)

EDM:n hyöty on siinä, että tietokannassa tiedon jakautuminen moniin tauluihin voi tuottaa ongelmia ohjelmistojen kehittäjille, koska tällöin tarvitaan monimutkaisia kyselylausekkeita. Tietokantaa suunniteltaessa tärkeää on huomioida ylläpidettävyys, turvallisuus, suorituskykyisyys ja skaalautuvuus. Kun tieto on organisoitu täyttämään äskeiset vaatimukset hyvästä tietokannasta, niin se tuottaa kehittäjille vaikeuksia päästä käsiksi tietoihin.

EDM:a käyttäessä kehittäjän ei tarvitse huolehtia tietokannan rakenteesta, taulujen nimistä, näkymistä tai tallennettujen toimintosarjojen nimistä tai niiden vaatimista parametreista.

> *Entity Data Model on konsepti. Entity rajapinta on erityinen toimeenpano, joka on realisoitu EDMX-tiedostona suunnitteluvaiheessa. Ajon aikana EDMX-tiedosto hajotetaan kolmeen erilliseen XML-tiedostoon. EDM viittaa konseptiin käyttää mallia kuvastamaan entiteettijä sovelluksessa.* (Lerman 2010, 20.)

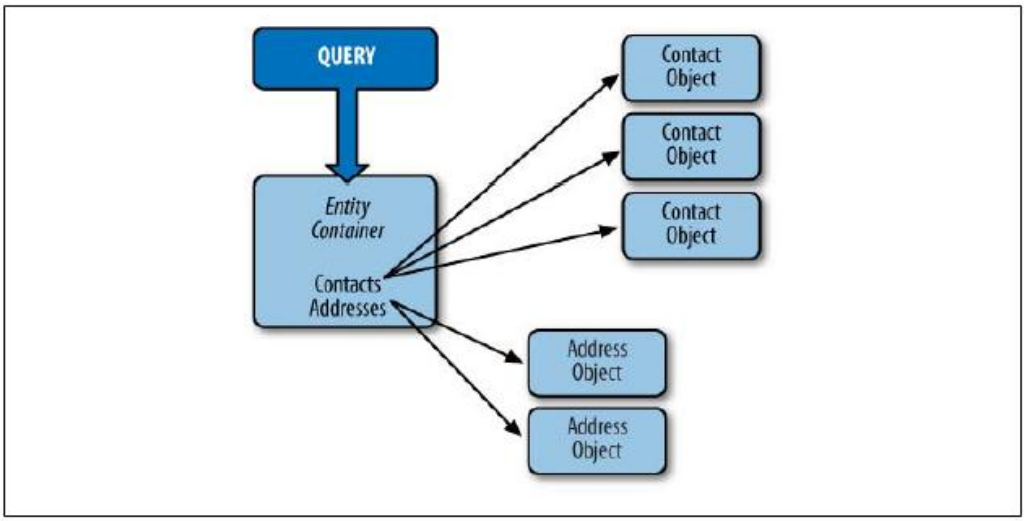
Entity Frameworkissa EDM:n toimeenpanon ensisijainen XML-tiedosto (CSDL) kuvastaa mallin käsitettä, joka on itseasiassa juuri EDM. Toinen XML-tiedosto (SSDL) kuvastaa tietokannan skeemaa ja kolmas (MSL) kuvastaa näiden kahden keskenäistä kartoitusta. Nämä XML-tiedostot ovat eriteltynä kuviossa 14. (Lerman 2010, 19 - 21)



KUVIO 14. Entity Data Model -rakenne (Lerman 2010, 29)

5.3 Entiteettisäiliö

Entiteettisäiliö on kääre entiteetti- ja assosiaatiojoukoista. Assosiaatiojoukot referoivat assosiaatioita entiteettien välillä. Entiteettisäiliö on kriittinen tulokohta tehtäessä kyselyitä malliin. Se paljastaa entiteettijoukot, ja kyselyt tehdään niitä vasten. Entiteettijoukkojen kautta pääsee käsiksi yksittäisiin entiteetteihin (kuvio 15). (Lerman 2010, 34.)



KUVIO 15. Entiteettisäiliön suhteet entiteettijoukkoihin ja entiteettiobjekteihin (Lerman 2010, 34)

5.4 Entiteettijoukko

Entiteettijoukko on säiliö entiteettityypeille. Sen kaksi attribuuttia ovat nimi (name) ja entiteettityyppi (entity type). Entiteettityyppi määrittää, mitä entiteettejä entiteettijoukko sisältää, ja määrittää joukolle nimen käyttäen nimi-attribuuttia. Nimi määritellään yleensä kuvastamaan entiteettijoukon sisältöä. Entiteettijoukon rakenne lähdekoodissa on näkyvillä kuviossa 16. (Lerman 2010, 35.)

```
<EntitySet Name = "Addresses"
    EntityType="SampleModel.Address" />
<EntitySet Name = "Contacts"/>
```

KUVIO 16. Entiteettijoukko

5.5 Entiteettityyppi

Entiteettityyppi on tietotyyppi mallissa. Entiteettityypin XML-skeemassa olennaisia elementtejä ovat avain (key) ja ominaisuus (property):

```
<EntityType Name = "Address">
  <Key>
    <PropertyRef Name = "addressID" />
  </Key>
  <Property Name="addressID" Type="Int32" Nullable="false"
  toreGeneratedPattern="Identity" />
  <Property Name="Street1" Type="String" MaxLength="50"
  Unicode="true" FixedLength="true" />
  <Property Name="Street2" Type="String" MaxLength="50"
  Unicode="true" FixedLength="true" />
  <Property Name="City" Type="String" MaxLength="50"
  Unicode="true" FixedLength="true" />

  <NavigationProperty Name="Contact"
  Relationship="SampleModel.FK_Address_Contact"
  FromRole="Address" ToRole="Contact" />

</EntityType>
```

KUVIO 17. Entiteettityyppi

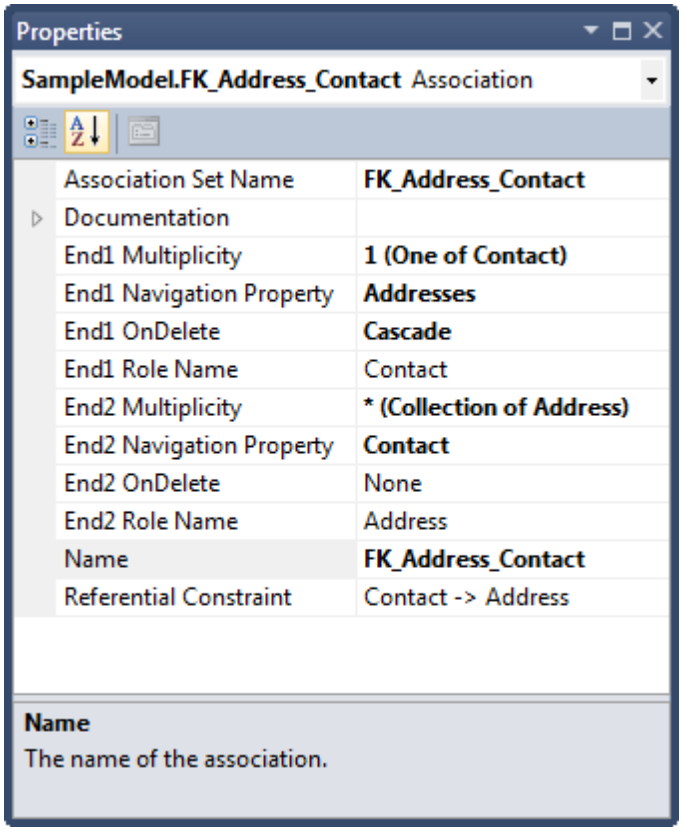
Avainarvo määrittää ominaisuuden ja muodostaa identiteettiavaimen entiteetille. Entiteetin avain on tärkeässä osassa entiteetin elinkaarta, ja se esimerkiksi mahdollistaa ohjelman seuraamaan entiteettiä ja suorittamaan tietokannan päivitykset ja uudelleenlataukset. Kuviossa 17 avainarvo on Address-entiteetille yksittäinen ominaisuus, addressID. Avainarvo voi kuitenkin muodostua useammasta avainominaisuudesta. Näitä kutsutaan entiteettiyhdistelmäavaimiksi, ja ne ovat samankaltaisia kuin yhdistelmäavaimet tietokannoissa. (Lerman 2010, 36.)

Ominaisuuselementeillä on määriteltynä nimi, mutta sen lisäksi määrityksissä on annettu sen tietotyyppi ja monia muita ominaistyyppejä, jotka kuvastavat siitä. Tietotyyppejä, jotka määrittelevät näitä ominaisuuksia, kutsutaan yksinkertaisiksi tyypeiksi (simple type). Nämä ovat primitiivisiä tyyppejä entiteetti-rajapinnan objektimallissa ja ovat osittain verrannollisia .NET-rajapinnan tietotyyppeihin. Entiteetti-rajapinnan primitiiviset tietotyypit kuitenkin vain kuvastavat entiteettien ominaisuuksia. Tietotyypeillä ei ole omia ominaisuuksia. (Lerman 2010, 37.)

Navigaatio-ominaisuudet ovat sidoksissa assosiaatioihin, jotka edustavat yhteyttä eri entiteettien välillä.

5.6 Assosiaatiot

Assosiaatiot määrittelevät yhteyksiä ja suhteita eri entiteettityyppien välillä. Assosiaatiot eivät määrittele näitä kokonaan, vaan assosiaatiot määrittävät päätepisteet ja niiden monimuotoisuuden. Aiemmassa kuviossa 17 oli vain yksi assosiaatio, joka oli entiteettityyppien ’Contact’ ja ’Address’ välillä. Assosiaatioiden nimet tulevat tietokannan määritellyistä yhteyksistä taulujen välille.



KUVIO 18. Visual Studion ominaisuusnäkymä assosiaatiolle (Lerman 2010, 39)

Kuviossa 18 näkyy assosiaation ominaisuuksia. Assosiaatio listaa molemmat päätepisteet. Pääte1 (End1) on Contact-entiteettityyppi ja sen roolinimeksi on valittu ’Contact’, johon voidaan viitata muualta mallista. Pääte1:n arvot kertovat sen muista ominaisuuksia. Sen monimuotoisuus on määritelty siten, että sillä on vain yksi yhteyspiste. Navigaatio-ominaisuus osoittaa sen toisen päätepisteen entiteettityypin. Toisena päätepisteenä on Address-entiteettityyppi, ja sen arvoista voidaan nähdä

sen omaavan useamman yhteyspisteen näiden kahden välillä. (Lerman 2010, 39.)

6 VALVONTASOVELLUKSEN TOTEUTUS

Valvontasovellus suunniteltiin ja toteutettiin CGI Suomi Oy:lle. Työn tarkoituksena oli toteuttaa kokonaisuus, jonka avulla päivittäisvalvontoja saataisiin yksinkertaisettua ja nopeutettua. Valvontasovellus on suunniteltu Windows-käyttöjärjestelmä pohjalle, ja sen on tarkoitus pystyä tekemään valvonta Microsoft-työkaluilla rakennetuissa tietojärjestelmissä.

Valvonnat tehdään päivittäin ja niiden toteuttamiseen tarvitaan kaksi henkilöä. Valvonnat ovat asiakasympäristöstä riippuen erilaisesti suoritettavia ja henkilöt tulee kouluttaa valvontoja tekemään aina ympäristökohtaisesti. Opinnäytetyön tarkoituksena on tuoda kaikki valvonnat samalle raportille tarkastettavaksi, jolloin valvonnan suorittaminen nopeutuu huomattavasti ja suorittamisesta tulee yhdenmukainen ja uusien henkilöidenkin kouluttaminen valvontoja tekemään helpottuu. Raportoinnin avulla myös historiatiedon kerääminen valvonnoista on helpompaa, ja tällöin mahdollisiin toistuviin ongelmiin on helpompi puuttua.

Valvontaraportin ajatuksena on näyttää valvojalle tilanne asiakaskohtaisesti, jolloin valvojan on helppo heti nähdä kokonaiskuva päivän valvontatuloksista. Tällöin ongelmiin pystytään reagoimaan nopeasti eikä asiakasympäristöihin tarvitse kirjautua enää erikseen.

Valvontasovellus on toteutettu C#-kielellä Windows Form -muodossa ja sen tietovarastona toimii Microsoft Azure -pilvipalvelimella olevan virtuaalikoneen Microsoft SQL -tietokanta. Samalla pilvipalvelimella sijaitsee myös raportointikanta, jossa on valvontaraportit. Kokonaisuuden suunnittelu aloitettiin käymällä läpi nykyiset valvontametodit ja miettimällä, kuinka niitä voitaisiin vielä kehittää.

6.1 C# Windows Form -sovellus

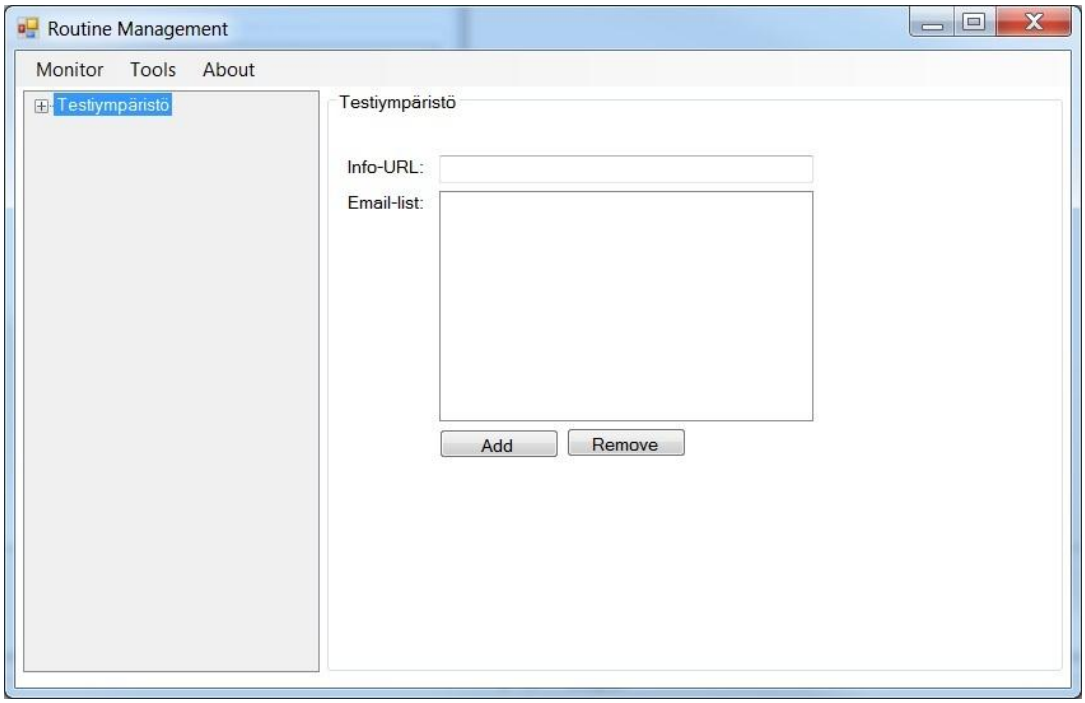
Sovellus on totetutettu luomaan valvontarutiineja eri asiakasympäristöihin. Sovellus ei itsessään toteuta valvontoja, vaan luo valvontarutiineista bin-tiedoston, joka viedään asiakasympäristössä olevalle erilliselle

sovellukselle, joka käsittelee bin-tiedoston ja tekee siihen määritellyt toimenpiteet ympäristössä ja palauttaa niiden tulokset Azure-pilvipalvelimen virtuaalikoneen tietokantaan.

Sovellus toteutettiin C#-kielellä, ja sen yhteys Azure-pilvipalvelimeen tapahtuu WCF-rajapinnan välityksellä. Pilvipalvelimella WCF-rajapinta on Entity Frameworkin välityksellä yhteydessä tietokantaan, josta se luo jokaista taulua kohti oman objektiluokan. Tietokantaan ei ole ulkoapäin suoraa yhteyttä, mikä lisää sinne tallennettujen rutiinien tietoturvaa.

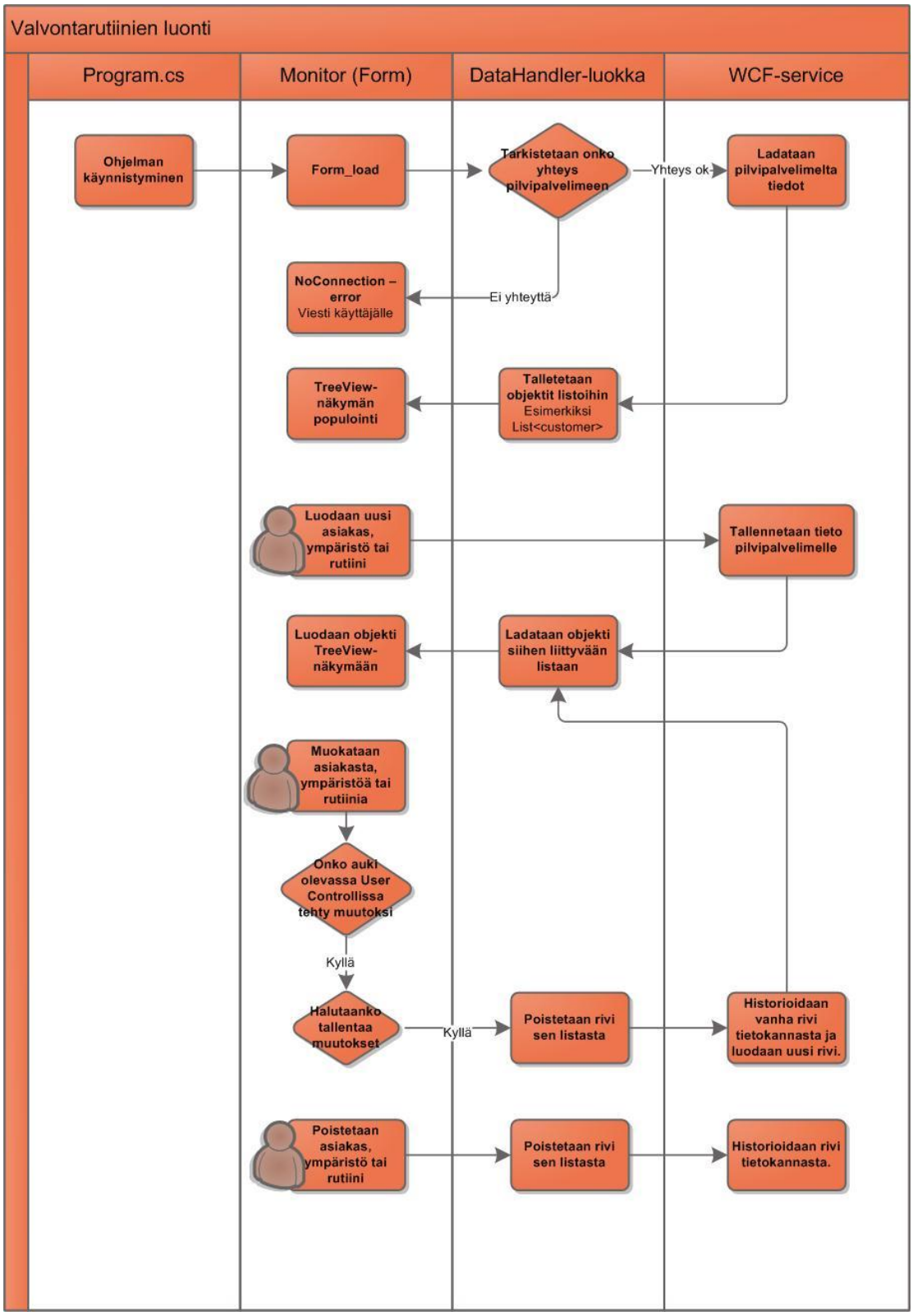
Sovellusta suunniteltaessa oli otettava huomioon, että sen oli oltava mahdollisimman helppokäyttöinen ja toteutettujen valvontarutiinen olisi tallennuttava siten, että vaikka sovellusta käytettäisiin eri koneelta, niin tiedot olisivat samat. Valvontarutiinit luodaan kerran, minkä jälkeen niihin tehdään harvoin muutoksia, joten erillistä kirjautumista ei sovellukseen tämän vuoksi tehty. Sovellusta ei suoranaisesti ole tehty kaupalliseen käyttöön, ja se on pelkästään käytössä CGI Suomi Oy:llä, joka toteuttaa valvonnat niihin asiakasympäristöihin, jotka ovat valvontapalvelun tilanneet.

Sovelluksen käyttöliittymä koostuu kahdesta osasta: TreeView-näkymästä ja User Control - näkymästä. TreeView-näkymästä valitaan, minkä asiakkaan valvontarutiineja halutaan muokata ja miltä osin. TreeView-näkymän valinnan mukaan ohjelma lataa User Control -näkymään oikean kontrollerin, johon käyttäjä voi määrittää haluamansa arvot. Kuvio 19 kuvastaa sovelluksen käyttöliittymänäkymää, jossa on yhden asiakkaan valvontarutiinit. Vasemmalla on TreeView-näkymä ja oikealla User Control



KUVIO 19. Sovelluksen yleisnäkymä

Sovellus on rakennettu yhden Windows Form -luokan ympärille, johon kutsutaan dynaamisesti rakentuvia User Control -elementtejä. TreeView-hierarkian taso määrä, mitä User Controllia kutsutaan. Sovelluksen tietojen hallinnasta sovelluksen sisällä vastaa DataHandler-luokka, jonka eri funktioita kutsutaan Windows Form -luokasta ja User Controlleista. DataHandler on Singleton-luokka, eli se luodaan vain kerran ajon aikana ja samaa instanssia kutsutaan kaikista näistä luokista. DataHandler toimii myös tiedon viejänä WCF-palvelulle, jonka päätepiste on määritelty ohjelmassa. WCF-palvelu tuo DataHandlerin käytettäväksi Entity Frameworkin Modelin, joka luo objektiluokat tietokannan rakenteesta. Kuviosta 20 käy ilmi aktiviteettikaavio valvontojen luonnista ja ohjelman eri luokkien toiminnoista eri tapahtumaketjuissa.

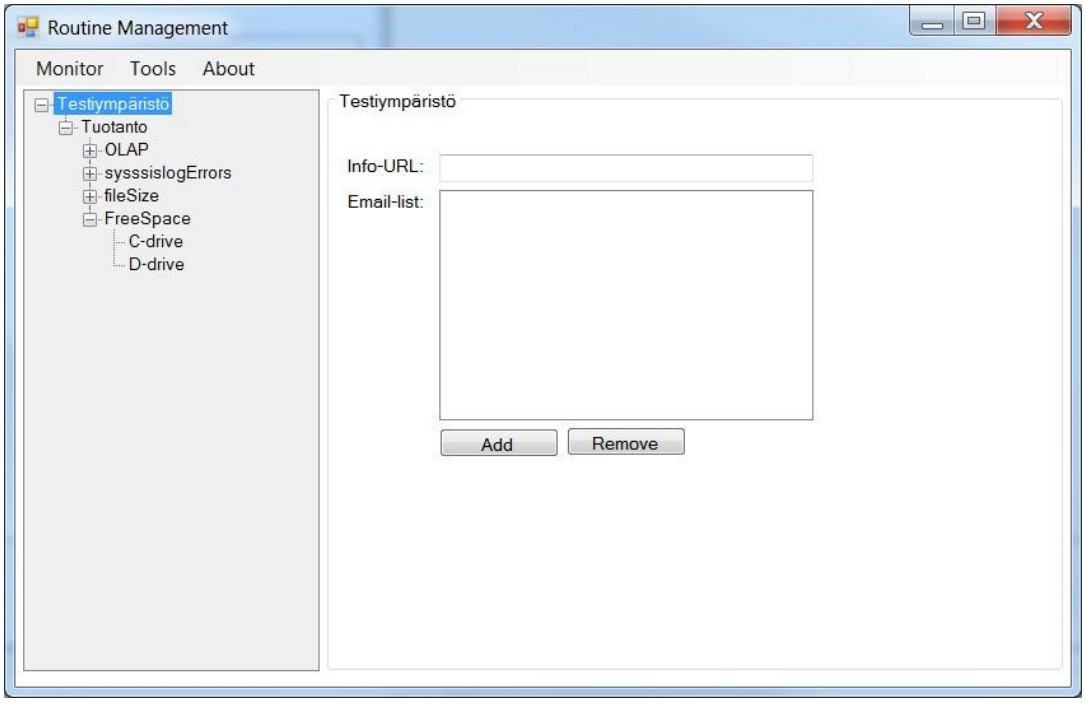


KUVIO 20. Aktiviteettikaavio valvontojen luonnista

6.1.1 Valvontarutiinin luonti sovelluksella

Jotta sovellusta on mahdollista käyttää, on käyttäjällä oltava verkkoyhteys Azure-pilvipalvelimeen, ja Azure-pilvipalvelussa olevan virtuaalikoneen on oltava käynnissä. Sovellus ei tallenna offline-tilassa valvontoja, vaan tallennus tapahtuu suoraan virtuaalikoneen tietokantaan. Offline-tallennusta mietittiin sovellusta tehtäessä, mutta siitä luovuttiin, koska valvontarutiinit tehdään yleensä kerran ympäristöä kohti ja niitä joudutaan muuttamaan harvoin. Sovellus antaa käyttäjälle virheviestin, mikäli yhteys virtuaalikoneeseen ei ole auki.

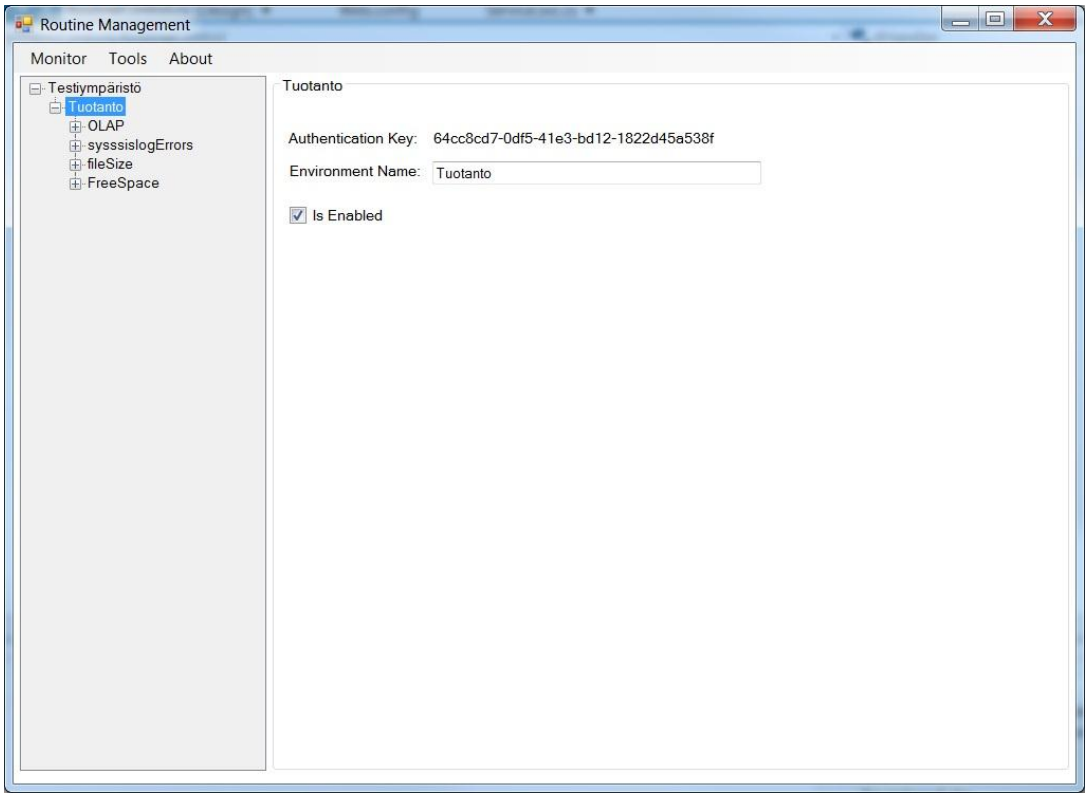
Valvontarutiinia luotaessa luodaan yksi TreeView-taso kerrallaan. Luonti tapahtuu oikealla hiiren klikkauksella, joka avaa valinnan joko lisätä TreeNode tai poistaa kyseinen kohta. TreeView toteuttaa asiakaskohtaisesti hierarkian:
Asiakas -> ympäristö -> käytettävä rutiini -> luotu rutiini
Kuvio 21 näyttää kuinka tämä hierarkia näkyy sovelluksessa.



KUVIO 21. Asiakkaan Testiympäristö TreeView-hierarkia

Yhtä asiakasta kohti voidaan luoda useampi ympäristö, koska asiakkaalla voi esimerkiksi olla tuotantopalvelimen lisäksi testipalvelin, jota tulee

valvoa, tai toinen tuotantopalvelin. Ympäristön User Control -näkymä on nähtävissä kuviossa 22, jossa näkyy ympäristön luodessa syntyvä autentikaatioavain. Tämän avulla varmistetaan, ettei valvontarutiiniin tehdä ulkoisia muutoksia. Ympäristöön valitaan tämän jälkeen halutut valvontarutiini, joiden valintavaranto on ennaltamääritelty sen mukaan, mitä ohjelma, joka toteuttaa valvonnan ympäristössä, osaa käsitellä. Valitut valvontarutiinit voidaan nimetä käyttäjän haluamalla tavalla, ja ne listautuvat kyseisen valvontarutiinin alle.



KUVIO 22. Tuotanto-palvelimen User Control -näkymä

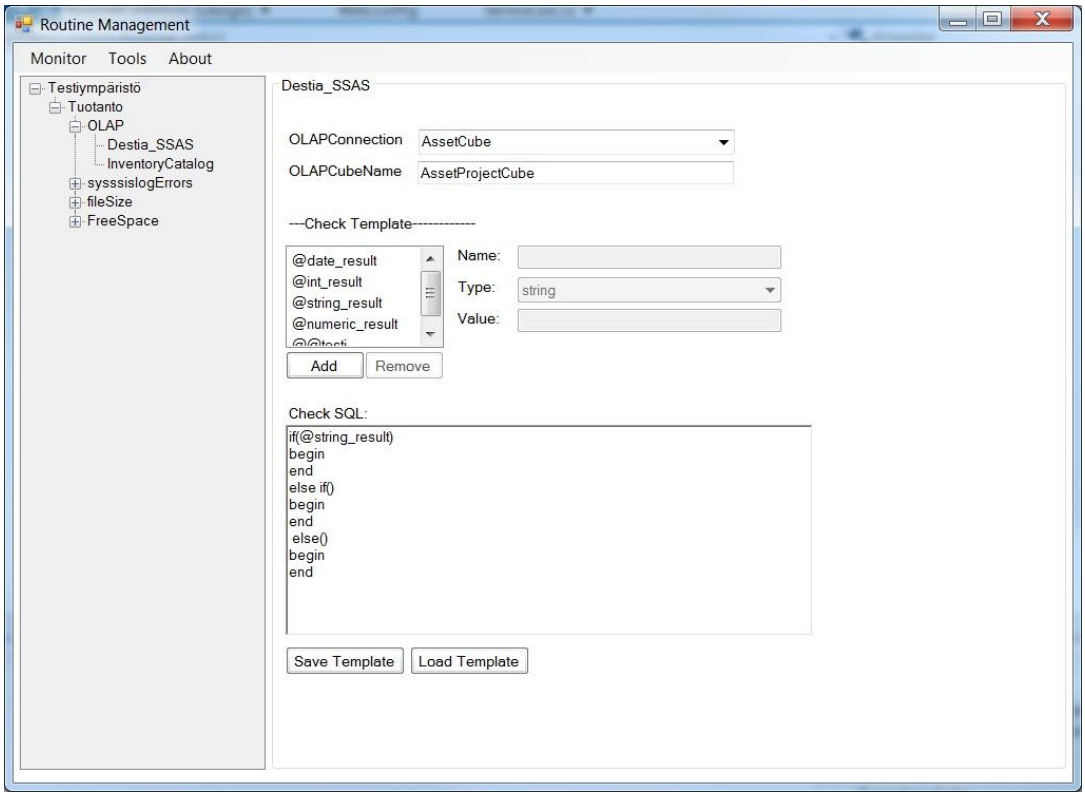
6.1.2 Rutiinin vertailuarvojen määrittäminen

Rutiineille voidaan luoda lisäksi vertailuarvoja, joita tarkastellaan ajojen läpiviennin jälkeen. Vertailuarvojen tarkastelu on toteutettu tallennetulla proseduurilla, joka ajetaan läpi tietokannassa ajastetusti. Vertailuarvojen avulla voidaan luoda virhe- tai varoitus-ilmoitus, mikäli esimerkiksi levytilan määrä on vähemmän kuin vertailuarvoksi määritelty tai mikäli ETL-ajojen

aikaleima on edelliseltä päivältä. Vertailuarvot määritellään rutiinin User Control -näkymään ’Check Template’ -kohdan alle. Vertailuarvot tehdään valvontatuloksille, joita on neljää eri muotoa: date-, int-, string- ja numeric-arvot. Rutiinista riippuen käytössä on yksi tai useampi näistä ja käyttäjä näkee painamalla TreeView-näkymästä rutiinin ylätasoa, mitä palautusarvoja kyseinen rutiini käyttää.

Valvontatuloksien muuttuja-arvot ovat oletuksena tallennettu jokaiselle rutiinille Check Templaten alle. Tämän lisäksi käyttäjä voi itse luoda muuttujia vertailukyselyyn, jolloin mikäli vertailuarvoa halutaan muuttaa, niin voidaan helposti vaihtaa vain muuttujan arvoa.

Vertailukysely (Check SQL) luodaan SQL-muodossa ehtolauseilla. Ne ajetaan tallennetulla proseduurilla läpi jokaiselle rutiinikohtaiselle valvontatulokselle, jotka ovat menneet onnistuneesti läpi, ja tulos tallennetaan valvontatuloksen sarakkeeseen, josta raportin on ne helppo lukea valvontaa tehdessä. Ennen vertailukyselyn toteutusta tallennettu proseduuri asettaa kyselyyn liittyvät muuttujat, jotta vertailu on mahdollinen. Halutessaan käyttäjä voi tallentaa vertailukyselyn pohjan ja hyödyntää sitä esimerkiksi toisen ympäristön rutiinin vertailuarvoa tehdessä. Kuviosta 23 on nähtävissä asiakasrutiinin parametrien ja vertailuarvojen määrittelynäkymä.



KUVIO 23. Rutiinin User Control ja vertailukysely-näkymä

6.1.3 Tietojen lataaminen ja tallentaminen

Tietojen lataamisessa ja tallentamisessa tietokantaan on käytetty hyödyksi WCF-rajapintoja ja Entity Frameworkia. Pilvipalvelimen päässä WCF-palvelu on määritetty IIS-webpalveluun (Internet Information Services) ja WCF-rajapinta on tuotu sovellukseen ServiceReferencenä, johon on määritelty osoite, jossa WCF-palvelun toinen päätepiste sijaitsee. Referenssiin on myös määritelty tietotyyppipalautus esimerkiksi kokoelma-joukolle. Sovellus luo käynnistyessään WCF-rajapinnan referenssin DataHandler-luokan käytettäväksi (Kuvio 24).

```
ServiceReference.ServiceClient serviceClient = new ServiceReference.ServiceClient();
```

KUVIO 24. ServiceReference

ServiceReferencen avulla voidaan kutsua pilvipalvelimen päässä olevan WCF-palvelun funktioita ja metodeita. Näillä metodeilla saadaan esimerkiksi tuotua WCF-palvelusta kerralla listallinen olio-objekteja tai vain tietty yksittäinen olio-objekti. Tiedon tallentaminen ja muokkaaminen WCF:n päässä toimii myös näiden metodien kautta.

WCF-rajapintaan on tuotu Entity Frameworkin avulla tietokannasta model-kuvaus, jolloin jokaisesta taulusta on luotu oma olio-objektinsa. Nämä olio-objektit on tuotu sovelluksen käyttöön lisäämällä ServiceReference myös kirjastona (kuvio 25).

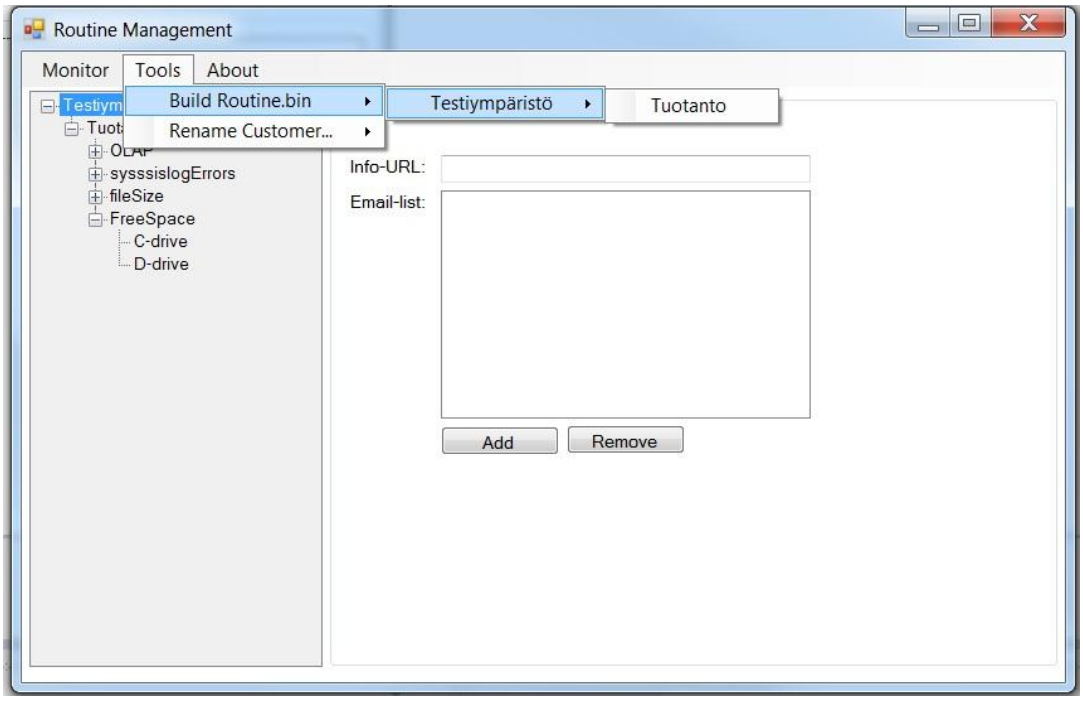
```
using CGIMonitorManagement.ServiceReference;
```

KUVIO 25. ServiceReference-kirjasto

Tiedon tallennus tapahtuu näihin olio-objektien muuttujiin, jolloin tiedot saadaan tallennettua suoraan tietokannan tarvitsemaan muotoon. Tietoa tallennettaessa kutsutaan WCF-palvelun metodia, jolle annetaan parametrina DataHandler-luokassa olio-objekti tai jossain tapauksissa lista olio-objekteja. Näiden avulla WCF-palvelun on yksinkertaista purkaa oliot pilvipalvelun päässä tietokantaan Entity Frameworkin avulla.

6.1.4 Valvontarutiini-tiedoston luonti

Asiakkaan ympäristöstä tehdyn valvontarutiinijoukon ollessa valmis, niin se tulee viedä valvonnan suorittavan ohjelman käytettäväsi asiakasympäristöön. Tätä varten siitä on luotava bin-tiedosto valvontasovelluksessa. Kuvio 26 kuvastaa polkua, jolla bin-tiedosto luodaan ohjelmassa. Bin-tiedostoa luotaessa valitaan ensiksi se, minkä asiakkaan rutiineille se halutaan luoda ja tämän jälkeen voidaan vielä kohdistaa valinta tiettyyn asiakasympäristöön. Ohjelmassa voidaan myös valita suoraan asiakas, jolloin sovellus luo kaikki siihen liittyvät ympäristöt.



KUVIO 26. Valvontarutiien bin-tiedoston luominen

Bin-tiedostoa luotaessa syntyy kaksi tiedostoa sovelluksen juureen: routines.bin ja config.xml. Routines.bin sisältää tiedot ajettavista rutiineista ja niiden tarvitsemista parametreista. Config.xml-tiedosto sen sijaan sisältää tiedon tietokanta- ja kuutiokantayhteyksistä, joita valvonnassa mahdollisesti tarvitaan. Näiden kantojen yhteyskonfiguraatioita ei tallenneta lainkaan pilvipalvelimelle, johon rutiinit tallennetaan, tietoturvallisuuden vuoksi. Bin- ja config-tiedostot viedään käsin asiakasympäristöön.

6.2 Azure-palvelin

Azure-pilvipalvelimen virtuaalikoneelle on asennettu Microsoft SQL Server, johon yhteys on rajattu vain virtuaalikoneelta tuleviin pyyntöihin. WCF:n ja Entity Frameworkin avulla saadaan tietoa kuljetettua edestakaisin sovelluksien välillä. Tämä lisää huomattavasti tietokannan tietoturvaa.

Azure-pilvipalvelussa oleva virtuaalikone kuuluu CGI Suomi Oy:n Microsoftilta vuokraamiin pilvipalvelimiin, ja sen käynnissä olemista hallinnoi erillinen virtuaalikone, joka vastaa kaikkien muidenkin koneiden

käynnistyksestä ja sammutuksesta. Näin virtuaalikoneiden käynnissä olemisaikaa saadaan rajoitettua ja niiden käytöstä saadaan kustannustehokkaampaa.

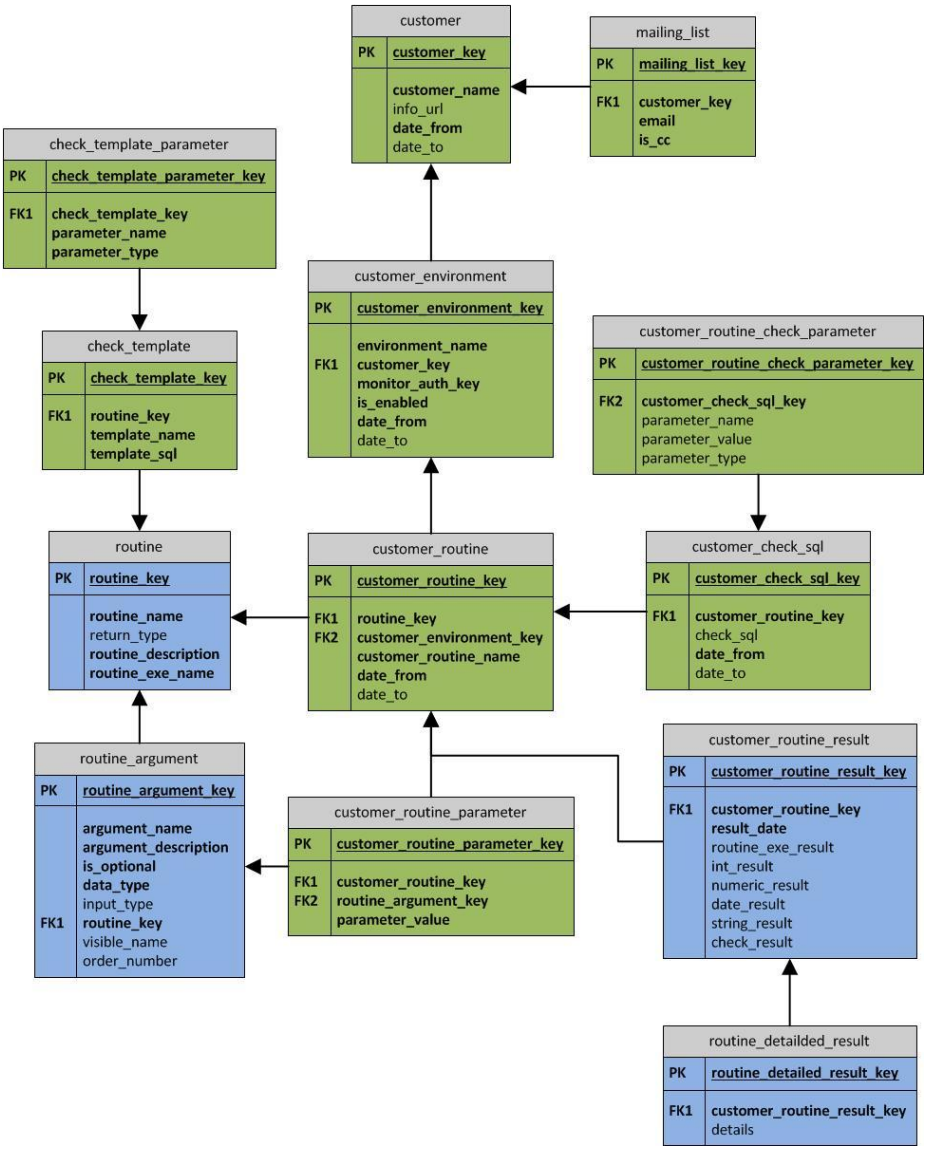
6.3 Tietokanta

Tietokanta on rakennettu käyttäen Microsoft SQL:ä, ja tiedot on jaettu riittävän useaan tauluun, jotta tietokanta on hyvin skaalatuva, muutosjoustava ja eheä. Tauluihin, johon sovelluksen luomia valvontarutiineja tallennettaan, on luotu historiointia varten DateFrom- ja DateTo-arvot. Kun valvontarutiini tallennetaan tauluun, se saa DateFrom-arvon senhetkisestä ajanhetkestä, ja kun arvo poistetaan tai sitä muokataan, niin sille annetaan DateTo-arvo. Tämän avulla raportointia voidaan historioida pidemmältä ajalta ja voidaan tarkastella myös vanhoja vertailuarvoja. Taulukossa 5 on kuvattu taulujen tarkoitus tietokannassa ja tiedon tallennuksessa.

TAULUKKO 5. Taulujen kuvaukset

Taulu	**Kuvaus**
customer	Kaikki asiakkaat, joille luotu valvontaympäristöjä.
mailing_list	Sähköpostiosoitteet, johon asiakkaan valvontaviesti lähetetään.
customer_environment	Asiakkaiden ympäristöt, joihin luotu valvontarutiineja.
routine	Käytössä olevat rutiniit, joita valvonnan tekevä ohjelma pystyy suorittamaan.
routine_argument	Rutiniinen tarvitsemat parametritiedot.
customer_routine	Asiakkaille määritellyt rutiinit.
customer_routine_parameter	Asiakkaille määritettyjen rutiinien parametrit.
check_template	Tallennetut pohjat vertailukyselyihin.
check_template_parameter	Tallennettujen pohjien muuttujat.
customer_check_sql	Vertailukysymykset asiakasympäristöjen rutiineille.
customer_routine_check_parameter	Vertailukysymysten muuttujat.
customer_routine_result	Valvontaprosessin tulokset.
routine_detailed_result	Valvontaprosessin tuloksien tarkemmat tiedot.

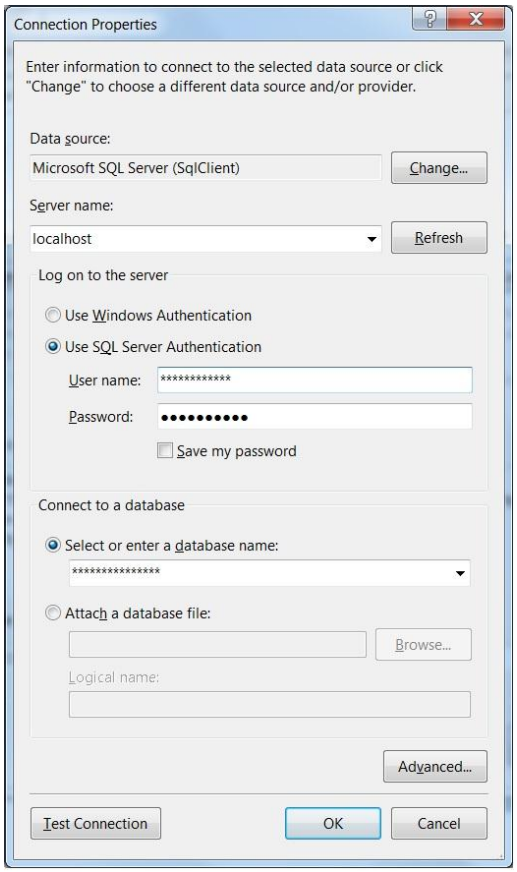
Sovellus, jolla rutiinit luodaan, täyttää osaa tauluista, mutta osa tauluista on valvonnan suorittaman sovelluksen käyttämiä. Routine- ja routine_argument-taulut on populoitu manuaalisesti, ja ne sisältävät tiedon siitä, mitä rutiineja asiakkaan päässä oleva valvontaohjelma osaa suorittaa. Sovellus myös tallettaa valvonnan tehtyään valvonnan tulokset customer_routine_result- ja routine_detailed_result-tauluihin. Kuviossa 27 on kuvattu tietokantakuvaus, jossa vihreällä värillä on merkitty rutiinin luomisesta vastaavan sovelluksen populoimat taulut ja sinisellä valvonnan suorittavaan sovellukseen liittyvät taulut.



KUVIO 27. Tietokantakuvaus

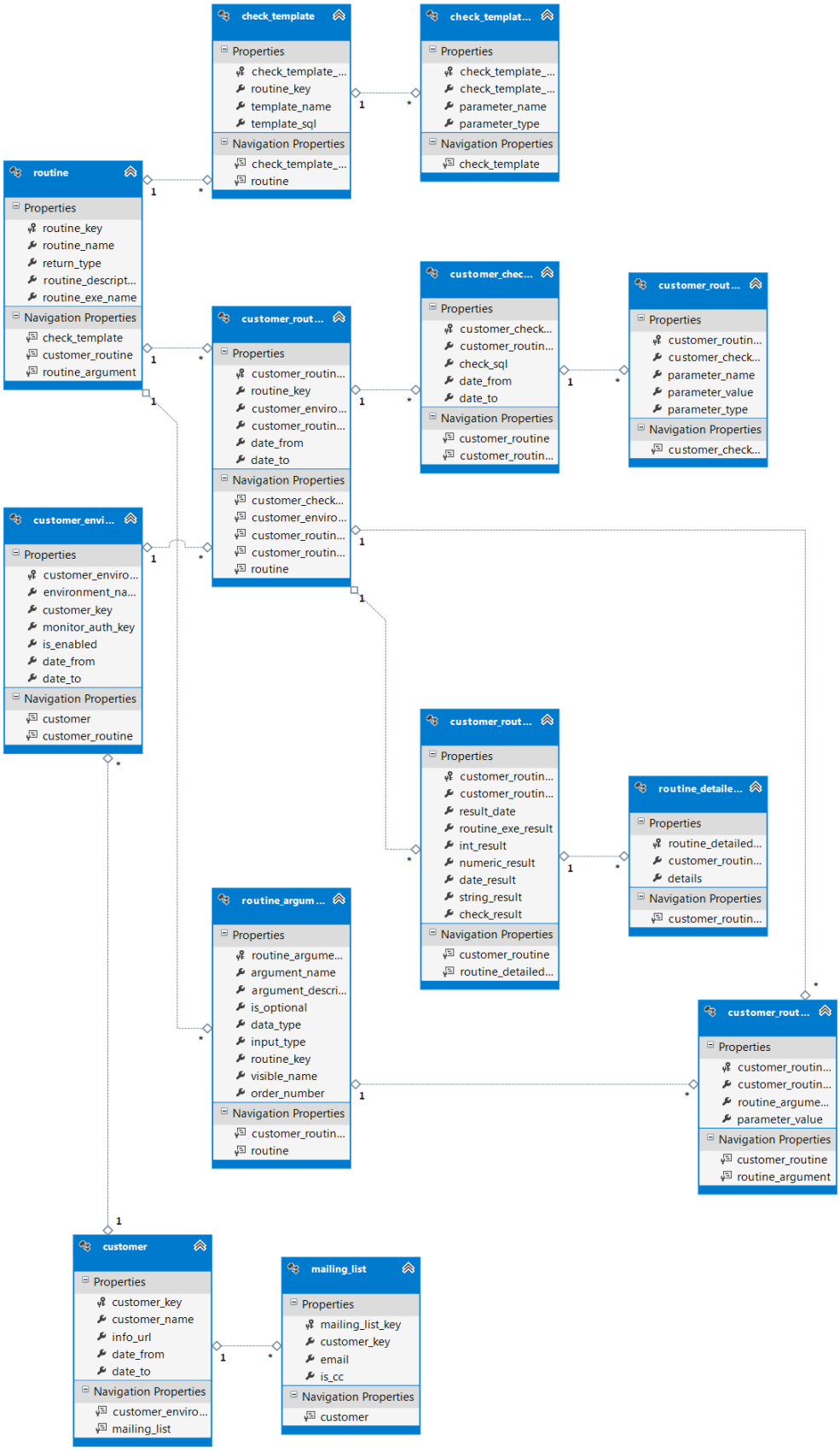
6.4 Entity Framework -rajapinta

WCF-palveluun on lisätty ADO.NET Entity Data Model, joka luo entiteettimallin käytettävästä tietokannasta WCF-palvelun käytettäväksi. Kuviossa 28 näkyy Data Modelin vaatimat tiedot tietokannasta. Käytettävä tietolähde on Microsoft SQL Server ja lähdepalvelin on ’localhost’, koska tietokanta on WCF-palvelun kanssa samalla palvelimella. Kirjautuminen palveluun on määritelty SQL Server -autentikoinnilla, jolloin kirjautumisen oikeuksia voidaan helposti rajoittaa. Tämän jälkeen valitaan vielä käytettävä tietokanta.



KUVIO 28. ADO.NET Entity Data Modelin yhdistäminen tietokantaan

Yhteyden luotua rajapinta hakee Data Modelin tietokannasta ja luo siitä EDMX-tiedoston, joka kuvastaa tietokannan rakennetta. Kuvio 29 kuvastaa pilvipalvelimen virtuaalikoneen WCF-palvelussa käytetyn Data Modelin tietokantaskeemaa.



KUVIO 29. Entity Frameworkin EDMX-tiedoston tietokantaskeema

6.5 Valvonnan suoritus

Valvonnat on määritelty ajastetusti alkamaan tiettynä ajanhetkenä, joka yleisesti ottaen on yöajojen jälkeen. Yöajoissa esimerkiksi ajetaan tietovarastoon tietoja ETL-ajoilla, tehdään jälkilaskentaa ja prosessoidaan kuutiot. Valvontaa suorittaessa asiakasympäristössä oleva sovellus lukee routines.bin- ja config.xml-tiedostot. Routines.bin sisältää kaikki rutiniit, jotka suoritetaan palvelimella ja config.xml tarvittavat yhteystiedot tietokantoihin ja OLAP-kantoihin. Valvonnat suoritettuaan sovellus ottaa yhteyden pilvipalvelimen WCF-palveluun ja lähettää valvontojen tulokset sen kautta tietokantaan. Tietokantaan on ajastettu tallennettu proseduuri, joka ajaa vertailukyselyt näille valvontojen tuloksille ja lisää vertailujen tulokset valvontariveille.

6.6 Valvonnan raportointi

Valvontojen raportointi tullaan toteuttamaan SSRS-raporttina (SQL Server Reporting Services). SSRS kuuluu Microsoft SQL Serverin työkaluihin. Raportti tulee sijaitsemaan pilvipalvelimen virtuaalikoneella, jolla tietokantakin on, ja se upotetaan iframe-tekniikalla verkkosivuun, jolloin sen tarkastelu ei vaadi kirjautumista palvelimelle, vaan autentikointi toteutetaan automaattisesti Windows-käyttäjätunnuksilla verkkosivulle mentäessä.

Raportti hakee valvontatiedot tietokannasta näkymien kautta. Oletuksena raportti hakee vain kyseisen päivän tiedot, mutta sen hakukriteereitä voidaan muuttaa, jotta saadaan suurempikin aikaväli haettua. Tämä on hyödyllistä esimerkiksi tarkasteltaessa viikonlopun jälkeen valvontoja. Raportilla näytetään päätasolla kaikkien asiakkaiden valvontojen tulokset liikennevalo-logiikalla:

- vihreä – kaikki ok
- keltainen – vertailuarvoista tullut huomautus
- punainen – valvonnassa havaittu virhe.

Päätasolta voidaan porautua asiakkaan ympäristöihin, josta nähdään liikennevalot asiakasympäristö kohtaisesti. Tästä voidaan porautua vielä ympäristössä ajettuihin rutiineihin, joista porautuminen vielä tarkempaan valvontatulokseen on mahdollista. Tällä porautumislogiikalla on nopea löytää, missä virhe on ollut asiakasympäristössä ja ilmoittaa virhetilanteesta asiakkaalle.

Asiakaille voidaan lähettää päivittäin raportti asiakkaan ympäristöjen valvonnoista. Tällöin SSRS-raportista saadaan tuotua HTML-sähköposti, joka toimitetaan asiakkaalle. Pilvipalvelimen virtuaalikoneen tietokantaan voidaan määritellä jokaista asiakasta kohti sähköpostiosoitteet, johon raportti halutaan lähettää. Raportti hakee sähköpostia lähettäessä nämä tiedot tietokannasta ja toimittaa sähköpostit sen mukaisesti.

7 YHTEENVETO

Tavoitteena valvontasovelluksen suunnittelussa ja toteutuksessa oli muodostaa automatisoitu ratkaisu päivittäisvalvontojen suorittamiseen ja helpottaa valvonnan suorittamista. Tarkoituksena oli luoda jokaista asiakasympäristöä kohtaan omat valvontarutiinit, jotka ajetaan ajastetusti asiakkaan ympäristössä. Valvonnoista oli tarkoituksena koota valvontatulokset ja historoida niitä, jolloin erilaiset toistuvat virhetilanteet saadaan karsittua pois. Tulokset oli tarkoitus esittää raporttipohjaisena, jolloin yhden henkilön on mahdollista suorittaa päivittäisvalvonta. Projektin läpivienti oli onnistunut ja asetetut tavoitteet saatiin suoritettua. Jatkokehitystä tarvitaan kuitenkin vielä valvontaraporttien luomisessa valvontatuloksista.

Valvontasovelluksen suunnittelu aloitettiin kesällä 2014 ja sen toteutus aloitettiin saman vuoden syksyllä. Aiemmin suunniteltu sovellus muuttui toteutuksen aikana hieman, kun sen staattisista ominaisuuksista luotiin enemmän dynaamisia. Suunnittelu oli keskittynyt myös enemmän toiminnallisuuksiin, joten käyttöliittymän suunnittelu oli jäänyt toissijaiseksi, mikä sitten heijastui virhetilanteina käyttöliittymän ja toiminnallisuuksien välillä. Sovelluksen testaus aloitettiin alkuvuodesta 2015. Testauksen aikana tuli ilmi useita virhetilanteita, varsinkin bin-tiedoston luomisessa oli korjattavaa, joita sitten korjattiin sovelluksesta testauksen aikana.

Luodessani valvontasovellusta käytin työkaluina Microsoft Visual Studiota ja SQL Serveriä. Valvontasovellus toteutettiin C#- ja Windows Form -toteutuksena, johon liitettiin sitten toiminnallisuuksia Windows Communication Foundationista ja Entity Frameworkistä. Sovellusta tehdessä työskenneltiin paljon Microsoft Azure -palvelimen toiminnallisuuksien parissa ja luotiin WCF-palvelu sen IIS-webpalveluun, johon sitten valvontasovellus on yhteydessä.

Valvonnan automatisoinnissa seuraava työtehtävä on luoda SSRS-raportit valvontatuloksista. Raportit oli tarkoitus toteuttaa alkuvuodesta 2015, mutta valvontasovelluksen aikataulun venymisen ja työresurssien puutteen

vuoksi toteutukselle ei ollut aikaa. Valvontaraportit on suunniteltu toteutettavaksi alkukesästä 2015.

LÄHTEET

ADO.NET Entity Framewrok At-a-Glance. 2015. [viitattu 17.3.2015]. Saatavissa: https://msdn.microsoft.com/en-us/data/aa937709

Kogent Solutions Inc. 2008. .NET Programming Black Book. New Delhi, Intia: DreamTech Press.

Lerman, J. 2010. Programming Entity Framework, 2nd Edition. Califoria, United States: O'Reilly.

Liu, M. 2012. WCF 4.5 Multi-Layer Services Development with Entity Framework, 3rd Edition. Birmingham, United Kingdom: Packt Publishing.

Löwy, J. 2010. Programming WCF Services. Califoria, United States: O'Reilly.

Microsoft. 2008. Improving Web Services Security. Saatavissa: https://wcfsecurityguide.codeplex.com/releases/view/15892

Overview of the .NET Framework. 2015. [viitattu 17.3.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/zw4w595w%28v=vs.110%29.asp

Pathak, N. 2011. Pro WCF4: Practical Microsoft SOA Implementation, 2nd Edition. New York, United States: Springer Science+Business Media.

Thai, T.L. Lam,H. 2002. .NET Framework Essentials, 2nd Edition. Califoria, United States: O'Reilly.

Windows Communication Foundation Architecture. 2015. [viitattu 17.3.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/ms733128.aspx